Universidad Zaragoza
1542

cnes
CENTRE NATIONAL D'ÉTUDES SPATIALES

Proyecto Fin de Carrera

Estudio de una aplicación domótica para la gestión de una vivienda vía satélite

Autor

Alexandra Ubau Rubio

Director

Jean-Claude Patalano

Ponente

Jesús Alastruey Benedé

Escuela de Ingeniería y Arquitectura

Zaragoza, Febrero de 2014

Agradecimientos

A Jean-Claude Patalano, a todo el departamento de RF/AR del CNES y a Jesús Alastruey Benedé por su ayuda, seguimiento y dedicación durante el desarrollo del proyecto.

A mi familia, especialmente a mis padres y a mi hermana por estar a mi lado en los momentos más difíciles a lo largo de todos estos años.

A mis amigos y compañeros de carrera por acompañarme durante todo este tiempo lleno de buenos y no tan buenos momentos.

Resumen

La proliferación de dispositivos tecnológicos para el hogar y su rápida evolución ha aumentado el interés de conectar todos ellos en red para integrarlos en un sistema de monitorización y control y poder gestionarlos de forma centralizada.

En este trabajo se estudia el control de una vivienda desde cualquier parte del mundo. En este contexto, el organismo gubernamental francés CNES (Centro Nacional de Estudios Espaciales), encargado del desarrollo espacial en Francia, propone el estudio del envío de la información necesaria para la gestión de una vivienda a través del enlace vía satélite. Es evidente que no se trata del medio óptimo para ello, pero hay casos en los que, aunque el retardo sea elevado, lo verdaderamente importante es que exista la posibilidad de comunicarse, siendo éste el objetivo último de los estudios que se realizan dentro del departamento de Radiofrecuencias y Aplicaciones Telemáticas en el que tuvo lugar mi estancia en dicha empresa.

El objetivo del proyecto es la puesta en marcha de una plataforma de pruebas compuesta por sensores, accionadores y dispositivos de control conectados a una red de manera que puedan ser controlados mediante un protocolo domótico y el posterior estudio de la viabilidad del satélite como medio de transmisión.

Para ello se estudiaron diferentes protocolos domóticos existentes en el mercado y se seleccionó uno adecuado para nuestro caso. Puesto que no se disponía de equipos domóticos reales, se eligieron una serie de dispositivos no domóticos (Arduino, indicadores luminosos, pulsadores,…) que fueron usados para establecer nuestra plataforma de pruebas. El siguiente paso fue la elaboración de diversos subsistemas que nos permitieron ir aumentando progresivamente en complejidad hasta llegar al sistema en el que quedaba establecida la red domótica. Y finalmente se llevo a cabo el escenario final en el que se integraba el satélite como medio de transmisión.

Durante este proyecto se han utilizado diversas plataformas y herramientas hardware y software como Arduino, para crear los periféricos domóticos necesarios, LabVIEW para simplificar la programación de las placas Arduino, Python, para crear menús interactivos para manejar dichas placas, Wireshark, para el análisis de los paquetes transmitidos a través de las redes y Netdisturb para simular las perturbaciones debidas al tránsito de la información a través del satélite.

Se trata de un proyecto que fue propuesto como el inicio de un largo estudio y se prevé que sea finalizado gracias a posteriores colaboraciones. Con dicho proyecto realizado durante mi estancia en el CNES, queda cubierto el objetivo inicial de establecer las bases para poder llevar a cabo este gran estudio.

Índice

Tabla de Figuras

Lista de acrónimos y abreviaciones

ADSL: Línea de abonado digital asimétrica (sigla del inglés *Asymmetric Digital Subscriber Line*) es un tipo de tecnología de línea DSL.

Atmega328: Chip microcontrolador creado por Atmel perteneciente a la serie megaAVR.

CM11: Módulo encargado de inyectar señales X10 en la corriente eléctrica, a la vez que está constantemente escuchando la corriente eléctrica para registrar lo que por ella circule.

CNES (*Centro Nacional de Estudios Espaciales*, en francés *Centre National d'Études Spatiales*) es un organismo gubernamental francés a cargo del desarrollo espacial nacional. Tiene su sede en París y fue fundado en 1961.

DHCP (sigla en inglés de *Dynamic Host Configuration Protocol*, en español «protocolo de configuración dinámica de *host*») es un protocolo de red que permite a los clientes de una red IP obtener sus parámetros de configuración automáticamente.

DMX512 a menudo abreviado como **DMX** (**D**igital **M**ultiple**X**), es un protocolo electrónico utilizado en luminotécnia para el control de la iluminación de espectáculos, permitiendo la comunicación entre los equipos de control de luces y las propias fuentes de luz.

EEPROM o **E²PROM** son las siglas de *Electrically Erasable Programmable Read-Only Memory* (ROM programable y borrada eléctricamente). Es un tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente.

EHS: European Home Systems Protocol, protocolo europeo de sistemas domóticos.

EIB: European Installation Bus, bus de instalación europeo.

Ethernet es un estándar de redes de área local para computadores con acceso al medio por contienda (CSMA/CD).

FTDI: Future Technology Devices International comúnmente conocido por su abreviatura, FTDI, es una compañía privada escocesa de dispositivos semiconductores, especializada en la tecnología (USB) de bus serie universal.

GPRS: General Packet Radio Service o servicio general de paquetes vía radio creado en la década de los 80 es una extensión del Sistema Global para Comunicaciones Móviles (Global System for Mobile Communications o GSM) para la transmisión de datos mediante conmutación de paquetes.

HTTP: Hypertext Transfer Protocol o HTTP (en español *protocolo de transferencia de hipertexto*) es el protocolo usado en cada transacción de la World Wide Web.

ICMP: El Protocolo de Mensajes de Control de Internet o ICMP (por sus siglas en inglés de *Internet Control Message Protocol*) es el sub protocolo de control y notificación de errores del Protocolo de Internet (IP).

IR: La radiación infrarroja, o radiación IR es un tipo de radiación electromagnética y térmica, de mayor longitud de onda que la luz visible, pero menor que la de las microondas.

IP: Internet Protocol (en español 'Protocolo de Internet') o IP es un protocolo de comunicación de datos digitales clasificado funcionalmente en la Capa de Red según el modelo internacional OSI.

KNX es un estándar (ISO/IEC 14543) de protocolo de comunicaciones de red, basado en OSI, para edificios inteligentes (domótica e inmótica). KNX es el sucesor de, y la convergencia de, tres previos estándares: el European Home Systems Protocol (EHS), BâtiBUS, y el European Installation Bus (EIB or Instabus).

MISO: (Master In Slave Out) Línea esclava que permite al maestro enviar los datos en el microcontrolador Arduino.

MOSI: (Master Out Slave In) Línea maestra para enviar los datos hacía los periféricos.

OpenWebNet es un protocolo de comunicación proyectado y desarrollado por BTicino a partir del 2000 que nace para permitir la interacción con todas las funciones disponibles en el sistema Domótico MyHome realizado por Bticino.

PLC: Power Line Communications es un término inglés que puede traducirse por comunicaciones mediante cable eléctrico y que se refiere a diferentes tecnologías que utilizan las líneas de energía eléctrica convencionales para transmitir señales de radio para propósitos de comunicación.

PWM: La modulación por ancho de pulsos, siglas en inglés de *pulse-width modulation*) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica.

RF: El término radiofrecuencia, también denominado espectro de radiofrecuencia, se aplica a la porción menos energética del espectro electromagnético, situada entre unos 3 kHz y unos 300 GHz.

RFID (siglas de *Radio Frequency IDentification*, en español identificación por radiofrecuencia) es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas, transponedores o tags RFID.

RFXcom: empresa que fabrica dispositivos, entre ellos de uso domótico.

RJ-45 (registered jack 45) es una interfaz física comúnmente usada para conectar redes de cableado estructurado. Posee ocho pines o conexiones eléctricas, que normalmente se usan como extremos de cables de par trenzado.
SCK: (Serial Clock) Reloj del microcontrolador Arduino que permite la sincronización en la transmisión de datos.

SMS: El servicio de mensajes cortos, servicio de mensajes simples o (Short Message Service) es un servicio disponible en los teléfonos móviles que permite el envío de mensajes cortos (también conocidos como mensajes de texto, o más coloquialmente, textos) entre teléfonos móviles que inventó un finlandés, Matti Makkonen junto al GMS en 1985.

SPI: El Bus SPI (del inglés Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.

SRAM Static Random Access Memory, o Memoria Estática de Acceso Aleatorio es un tipo de memoria que, a diferencia de la memoria DRAM, es capaz de mantener los datos mientras esté alimentada, *sin necesidad de circuito de refresco*.

SS: (Slave Select) Pin que permite en el microcontrolador Arduino activar o desactivar dispositivos específicos.

TCP: Transmission Control Protocol (en español 'Protocolo de Control de Transmisión') es uno de los protocolos fundamentales en Internet.

TWI o I²C es un bus de comunicaciones en serie. Su nombre viene de Inter-Integrated Circuit (Inter-Circuitos Integrados). La velocidad es de 100 kbit/s en el modo estándar, aunque también permite velocidades de 3.4 Mbit/s. Es un bus muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados (Embedded Systems) y generalizando más para comunicar circuitos integrados entre sí que normalmente residen en un mismo circuito impreso.

UDP: User Datagram Protocol es un protocolo del nivel de transporte basado en el intercambio de datagramas (Encapsulado de capa 4 Modelo OSI). Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera.

WAP: Wireless Application Protocol o (protocolo de aplicaciones inalámbricas) es un estándar abierto internacional para aplicaciones que utilizan las comunicaciones inalámbricas, p.ej. acceso a servicios de Internet desde un teléfono móvil.

Wi-Fi es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.

WiMAX siglas de Worldwide Interoperability for Microwave Access (interoperabilidad mundial para acceso por microondas), es una norma de transmisión de datos que utiliza las ondas de radio en las frecuencias de 2,3 a 3,5 GHz y puede tener una cobertura de hasta 50 km.

WPAN: Wireless Personal Area Network, Red Inalámbrica de Área Personal es una red de computadoras para la comunicación entre distintos dispositivos cercanos al punto de acceso. Estas redes normalmente son de unos pocos metros y para uso personal.

xPL es un protocolo abierto creado para permitir el control y monitorización en un sistema domótico.

X10 es un protocolo de comunicaciones para el control remoto de dispositivos eléctricos, que utiliza la línea eléctrica (220V o 110V) preexistente, para transmitir señales de control entre equipos de automatización del hogar (domótica) en formato digital.

X2D: Protocolo domótico creado por Delta Dore.

Z-Wave es un protocolo de comunicaciones wireless usado en domótica, especialmente para el control remoto de aplicaciones.

1-Wire es un protocolo de comunicaciones en serie diseñado por Dallas Semiconductor. Está basado en un bus, un maestro y varios esclavos de una sola línea de datos en la que se alimentan.

Capítulo 1. Introducción

1.1 Contexto del proyecto y motivación

A medida que pasa el tiempo, van apareciendo más y más dispositivos electrónicos que, junto con una unidad de control central, podrían ser conectados en red para gestionar la vivienda de una forma inteligente. Pero este concepto de vivienda inteligente es más complicado de implementar si todos estos dispositivos están aislados entre sí. Esta nueva filosofía aplicada al sector doméstico, en la que los sistemas actuales integran automatización, informática y nuevas tecnologías de la información, es la que implica la creación del neologismo domótica.

Así pues, la domótica consiste en la automatización de los aparatos del hogar para mejorar nuestra vida cotidiana, proporcionando servicios como seguridad, protección, comunicación a distancia, gestión energética, uso más racional de la energía, etc.

Uno de los puntos clave de la domótica es el intercambio de la información, puesto que permite conocer el estado de la red. Para transmitir la información generada por los equipos domóticos existen muchas formas de transmisión, como las ondas radio (WiFi), infrarrojos, comunicaciones mediante cable eléctrico (PLC), *Ethernet*,... Pero los límites de la transmisión por ondas radio y la necesidad de superar las barreras geográficas tales como montañas, condujeron al desarrollo de la tecnología satelital para la transmisión de mensajes.

En el caso de las posibles aplicaciones domóticas, la idea es permitir a una familia la gestión de su vivienda desde lugares donde se carece de cobertura telefónica para acceder a internet. Por diversos motivos, ya sean laborales o de ocio, podemos encontrarnos en un país de África o en las islas del Pacífico. La idea básica sería poder conectarnos a nuestra vivienda desde cualquier sitio remoto a través del satélite como podemos ver en el siguiente esquema.

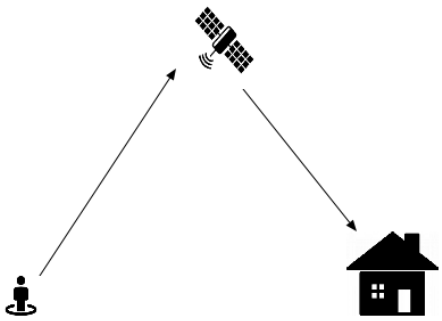

Figura 1. Esquema básico

Este proyecto surge en el departamento RF/AR (Departamento de Radiofrecuencias y Aplicaciones Telemáticas) del organismo gubernamental francés a cargo del desarrollo espacial nacional, CNES. En dicho departamento, sea cual sea la materia o el tema a tratar en cada uno de sus proyectos, se elige siempre el satélite como medio de transmisión, puesto que su misión es buscar nuevas aplicaciones de tecnología espacial, hecho que es posible gracias al acceso que tienen a los satélites y la parte del ancho de banda que se reserva de estos para las pruebas y estudios pertinentes.

Por tanto, aunque a priori el uso del satélite como medio de trasporte para enviar la información necesaria para el manejo y control de un sistema domótico no sea el más eficaz, debido a su gran coste y a otros factores como la necesidad de instalar antenas para recibir la señal, su gran retardo… en este caso, prima la posibilidad de comunicarse aunque no sea el medio óptimo, siendo esta la razón por la cual se propuso este proyecto.

En nuestro caso más concreto, el objetivo podría ser reducir la conocida brecha digital que existe en la región de Aragón, sobre todo por la presencia de los Pirineos y cuya solución se basa en dos grandes alternativas: las conexiones inalámbricas (WiMax) o las conexiones vía satélite.

1.2 Objetivos

Los objetivos principales del proyecto son verificar la capacidad del satélite y la viabilidad o los posibles problemas e impedimentos a la hora de transmitir la información necesaria para el funcionamiento de una casa "inteligente", identificar nuevas aplicaciones posibles gracias a las características de las comunicaciones vía satélite y validar el principio.

Para lograr estos objetivos, las tareas propuestas fueron:

- Identificar las áreas de aplicación en domótica para las cuales el satélite puede ser utilizado.
- Estudiar el estado del arte de los distintos sistemas, dispositivos, redes y protocolos domóticos existentes en el mercado.
- Diseñar una plataforma de pruebas compuesta por sensores, actuadores, dispositivos de control y redes controladas domóticamente que podrían insertarse en el segmento satelital.
- Desarrollar las pruebas y ensayos, análisis, extracción y síntesis de los resultados obtenidos.
- Analizar y obtener las conclusiones de los resultados obtenidos.

1.3 Organización del documento

El documento está organizado de la siguiente forma:

- El primer capítulo está dedicado a la introducción del documento.
- El segundo capítulo se centra en el estudio del estado del arte.

- El tercer capítulo está centrado en la definición y explicación del protocolo xPL, elegido para el desarrollo de este proyecto.
- En el cuarto capítulo se describen todas las herramientas y programas utilizados para la puesta en marcha del proyecto.
- El quinto apartado se centra en el trabajo experimental realizado durante los seis meses que duró mi estancia en el CNES, el cual se ha dividido en diferentes subsistemas hasta llegar al escenario final. También se exponen en este apartado los resultados correspondientes.
- En el sexto capítulo se exponen las conclusiones que se obtienen después de la finalización del proyecto.
- Y la séptima y última parte está compuesta por los anexos que amplían detalles sobre los contenidos del documento principal.

Capítulo 2. Estado del arte

La rápida aparición y evolución de las redes domésticas ha causado la creación de un cierto número de tecnologías y de protocolos, algunos de utilización específica en el hogar y otros heredados de la industria. En un primer momento, las entidades intentaban imponer sus propias iniciativas bus, pero más tarde se vieron obligadas a crear protocolos estandarizados para facilitar la introducción en el mercado de los productos domóticos. Fue a partir de entonces, cuando empezaron a surgir protocolos y normas domóticas como KNX, X10, EHS, X2D, Z-Wave, Openwebnet, xPL… En el presente capítulo presentaremos los diferentes tipos de tecnologías utilizadas en las comunicaciones y los soportes físicos apropiados para el uso en viviendas.

2.1 Los soportes de transmisión

El transporte de la información necesita de un medio de transmisión, que puede ser físico (por cable) o por aire (inalámbrico).

2.1.1 Por cable

- Coaxial: transmisión de señales eléctricas a alta frecuencia.
- Par cruzado: conexión en la que dos conductores son entrelazados para tener menos interferencias, aumentar la potencia y limitar la diafonía entre cables adyacentes.
- Fibra óptica: generalmente utilizadas en redes de datos, la información es enviada en forma de impulsos luminosos.
- PLC (Power Line Communications): para las tecnologías que utilizan las líneas eléctricas clásicas para transmitir datos con el objetivo de comunicar.

2.1.2 Inalámbricas

- Wi-Fi: envío de datos que utilizan las ondas radio.
- GPRS (General Packet Radio Service): extensión de Global System for Mobile Communications o GSM, para la gestión del envío de paquetes.
- Bluetooth: especificación industrial para redes inalámbricas WPAN que permite la transmisión a distancias cortas de la voz y de datos entre diferentes aparatos a través de un enlace radio en la banda ISM de 2,5 GHZ.
- Radiocomunicación por microondas y WiMAX: funcionan en la banda situada entre 1 a 40 GHz y 2 a 66 GHz respectivamente, donde las informaciones pueden ser transmitidas utilizando una onda portadora proveniente de un generador situado en una antena.
- Infrarrojo: radiación electromagnética en la que la longitud de onda es de 700 nanómetro a 1 micrómetro.
- ZigBee: nombre de un conjunto de protocolos de alto nivel de comunicación digital inalámbrica y de baja potencia. Está dirigida a las aplicaciones que necesitan una comunicación segura para poca cantidad de datos y una vida larga de sus baterías.

2.2 Tecnologías de transmisión y normas

Las tecnologías de transmisión son los protocolos o las reglas que los dispositivos de comunicación que deseen comunicarse deben comprender y utilizar para permitir la conexión, la comunicación y la transferencia de datos.

Los estándares son un conjunto de recomendaciones y de normas que las empresas deben seguir con el fin de asegurar que sus productos puedan interactuar con otros periféricos de otras marcas. A continuación presentamos algunos de los protocolos más conocidos.

2.2.1 KNX

El protocolo estándar KNX (Konnex) se convirtió en la norma domótica estándar internacional ISO / IEC 14 543-3. Se trata de una norma europea establecida en 1987, que reagrupa actualmente más de 190 fabricantes como Hager, ABB, JUNG, SCHNEIDER, BOSCH, ELECTROLUX, ...
El desarrollo paralelo de KNX y EIB hace que los proyectos realizados en el pasado sigan siendo compatibles con los dispositivos más recientes.
El protocolo KNX se basa en el principio de BUS, todos los participantes KNX de un proyecto (pulsadores, actuadores ...) interactúan entre sí a través de este bus. A diferencia de otros protocolos de automatización, éste no funciona en modo maestro/esclavo, cada controlador es independiente de los otros.

2.2.2 X10

La norma X10 es el estándar creado hace más de 20 años por la sociedad Powerhouse para controlar los electrodomésticos en los Estados Unidos y en el resto del mundo. Al igual que cualquier arquitectura de automatización tradicional del hogar, el estándar X10 utiliza la tecnología PLC para construir su red. El principio consiste en tener los emisores y los receptores conectados a la red eléctrica y comunicarlos entre sí.

2.2.3 EHS

El "European Home Systems" es un protocolo domótico de comunicación abierto que utiliza el PLC. La especificación EHS se ha definido de manera que los aparatos puedan comunicarse y compartir recursos entre ellos. El protocolo EHS se basa en un sistema de comunicación común y en definiciones inequívocas de la funcionalidad del dispositivo. El modelo de comunicación EHS sigue la estructura del modelo OSI. El protocolo EHS especifica la capa física, la capa de enlace de datos, la capa de red y la capa de aplicación.

2.2.4 X2D

El protocolo X2D es un protocolo de comunicación desarrollado por Delta Dore desde hace más de veinte años. Utiliza la tecnología de comunicación mediante cable eléctrico o radio (868 MHz) y permite la comunicación y el control de los equipos eléctricos siguiendo un concepto de automatización. Utiliza las frecuencias radio 434 MHz y 868 MHz. Garantiza una transmisión con un alcance de radio de 200 a 300 metros en campo abierto. El uso de dos frecuencias completamente diferentes mejora la calidad de la transmisión. Dichas frecuencias se emiten al mismo tiempo para que la una o la otra sea recibida por la central. La frecuencia de 868 MHz no es perturbada por las emisiones continuas como: auriculares inalámbricos, WiFi, ...

2.2.5 Z-Wave

Z-Wave es un protocolo para la comunicación inalámbrica entre dispositivos electrónicos. Este protocolo tiene como características principales:

- Está principalmente destinado a la automatización del hogar.
- Es relativamente seguro.
- Es de doble sentido (cada componente es a la vez transmisor y receptor).
- Se utiliza en un sistema de red en malla.

Al igual que cualquier señal RF inalámbrica, el alcance de una señal Z-Wave está fuertemente influenciada por el entorno en el que se emite. Utiliza frecuencias radio para las comunicaciones y permite que dos componentes electrónicos Z-Wave discutan juntos para intercambiar información.

2.2.6 OpenWebNet

OpenWebNet es un protocolo de comunicación diseñado y desarrollado por BTicino desde el año 2000. Dicho protocolo fue creado para permitir la interacción entre todas las funciones del sistema de automatización del hogar "*MyHome*". Una evolución reciente ha permitido utilizar OpenWebNet para interactuar con cualquier sistema de automatización con el uso de pasarelas adecuadas.

2.2.7 xPL

xPL es un protocolo abierto creado para permitir el control y la vigilancia de los dispositivos de automatización del hogar. El objetivo principal de xPL es proporcionar un amplio conjunto de características conservando una estructura de mensaje específica. El protocolo incluye capacidades de auto-configuración que soportan una arquitectura totalmente "*plug-n-play*" esencial para asegurar una buena experiencia para el usuario final. Las comunicaciones entre las aplicaciones xPL en una red de área local (LAN) utilizan el puerto UDP 3865.

Capítulo 3. Protocolo xPL

Puesto que decidimos utilizar el microcontrolador Arduino para llevar a cabo nuestros experimentos creímos que el protocolo xPL podría ser una buena opción, ya que cuenta con una biblioteca específica para Arduino y existen muchos ejemplos que combinan el protocolo xPL con Arduino y están disponibles en la Web.

3.1 Definición del protocolo

Uno de los principios de la domótica es lograr que diferentes dispositivos de la vida cotidiana lleguen a comunicarse entre ellos para conseguir que el conjunto sea más inteligente, controlable y extensible. Partiendo de esta base, una de las dificultades a las que nos enfrentamos es la complejidad a la hora de interconectar las diferentes tecnologías existentes en un sólo sistema domótico.

Teniendo en cuenta que somos capaces de utilizar diferentes tecnologías (PLCBus, Zwave, x10...) pero que estas sólo tienen interés si se usan conjuntamente, necesitamos un medio que nos permita manejar y reunir toda esta información para que luego podamos utilizarlo en un sistema de automatización del hogar, sin tener que entrar en los detalles de implementación de cada tecnología.

El protocolo llamado xPL (eXtremely simPle protocoL) tiene como objetivo permitir precisamente el control y seguimiento de todos los equipos del hogar mediante un único lenguaje de comunicación.

En sus inicios el proyecto se llamaba XAP. Este sistema permite, mediante el intercambio formateado de texto con un patrón muy específico, comprender los diferentes eventos que se producen en la red doméstica y ejecutar comandos que actúen sobre los equipos domóticos.

En enero de 2003, Ian Lowe y Tony Tofts crearon un proyecto en paralelo llamado xPL. Dicho protocolo, que requiere una red IP clásica y XML, permite la autoconfiguración y autodescubrimiento de los equipos permitiendo una integración simple en la red.

3.2 Funcionamiento del protocolo

El principio del protocolo xPL es simple. Un software sirve de pasarela (*HUB*), cada equipo compatible con dicho protocolo transmite mensajes (eventos, estados u órdenes) hacia el *hub* y este a su vez lo retransmitirá hacia el resto de equipos conectados en la red.

Un posible ejemplo de una red xPL sería:

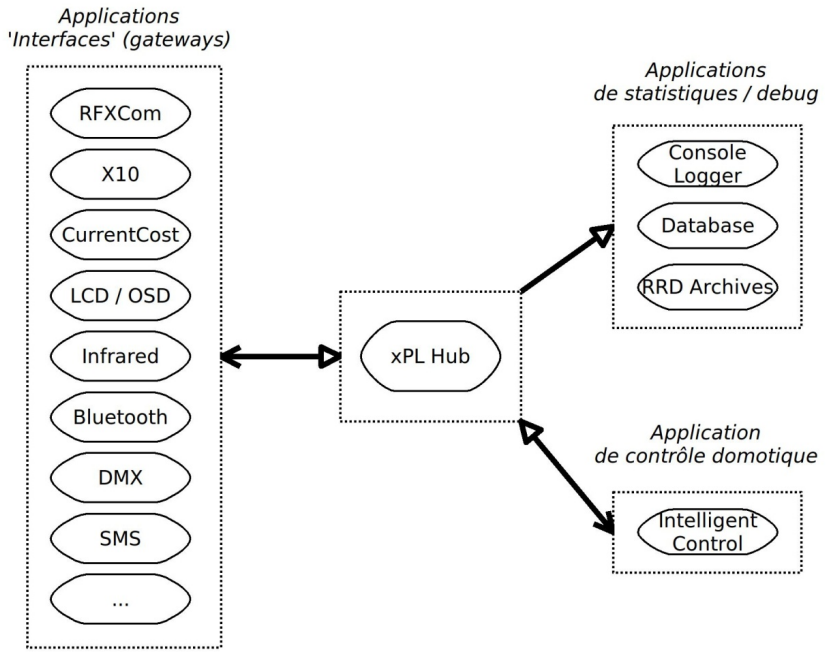

Figura 2. Esquema ejemplo de una red xPL

Cada óvalo en el diagrama anterior es una aplicación independiente que se encuentra en la mayoría de los casos instalada en un ordenador de mesa, pero pueden estar también integradas en otros equipos.

3.2.1 Interfaces (*Gateways*)

Las *Gateways* son aplicaciones que hacen de enlace entre las diferentes tecnologías y el protocolo xPL. Se trata de aplicaciones que conocen los detalles de implementación de las diferentes tecnologías. Por ejemplo encontramos puertas de enlace como las siguientes:

- X10 (que luego se conecta a la interfaz CM11 por ejemplo).
- RFXCom (que se conecta con el producto del mismo nombre y a continuación permite establecer una interfaz con todas las tecnologías conocedoras de RFXCom).
- CurrentCost (que se conecta a la consola de ENVI para medir el consumo eléctrico de la casa).
- SMS (que le permite enviar SMS).
- 1wire (utilizado para comunicarse con toda la gama de sensores 1Wire).
- ZWave (para comunicarse con la gama de automatismos ZWave).

3.2.2 Aplicaciones de estadística y de seguimiento

Estas son las aplicaciones que sólo reciben información y no emiten nada. Por ejemplo, podemos guardar todos los cambios de estado de los equipos de la casa en una base de datos o almacenar los datos provenientes de diferentes sensores para posteriormente analizarlos.

3.2.3 Aplicaciones de control

Se trata de las aplicaciones que contienen la inteligencia y toda la lógica del sistema domótico. Estas son por ejemplo las aplicaciones que van a decidir apagar la luz cuando es de día o bajar la calefacción cuando la casa esté vacía. Éstas reciben todos los datos provenientes de las aplicaciones (*gateways*) mediante el protocolo xPL y emiten sus órdenes hacia dichas pasarelas utilizando el mismo protocolo.

3.2.4 *Hub* xPL

Es el elemento indispensable en una red xPL. La presencia del *hub* es necesaria en cada uno de los equipos en los que se ejecutan una o más aplicaciones xPL. El *hub* no tiene inteligencia de control pero es él quien se encargará de encaminar los mensajes a sus aplicaciones xPL destino.

3.3 Aspectos técnicos

Las transmisiones en las que se basa el protocolo xPL se hacen en *broadcast* UDP. Es importante entender que dichas transmisiones no son un *broadcast* propiamente dicho como se conoce en telemática, sino que se le llama así porque el principio de funcionamiento del protocolo es el envío de paquetes a todos los equipos existentes en la red, de ahí el "*broadcast*", por lo que de ahora en adelante cada vez que aparezca dicho término en el documento será con esta definición particular.

Así pues, las aplicaciones emiten sus mensajes en la red y el *hub* los redistribuye a sus destinatarios. Por la propia definición del funcionamiento del protocolo, todas las aplicaciones reciben todos los mensajes incluidos los que ellas mismas han enviado. Gracias a este mecanismo, el *hub* mantiene una lista de las aplicaciones que están conectadas. Al recibir un mensaje de una nueva aplicación la registra en la lista para actualizar todas las aplicaciones existentes en la red en ese preciso instante.

3.4 Portabilidad

Es esta la característica que hace más potente al protocolo que es a su vez *open source* y sencillo. Hay aplicaciones xPL disponibles para los sistemas operativos más utilizados: *Windows, Linux, MacOS*. Por lo tanto, es posible tener diferentes sistemas operativos en la misma red xPL que podrán comunicarse entre ellos sin mayor problema gracias al protocolo.

3.5 Mensajes

Existen tres tipos de mensajes xPL:

- **xpl-cmnd** (control): Este mensaje permite enviar comandos de control al equipo xPL.
- **xpl-trig** (*trigger*): Este mensaje se genera automáticamente cuando el dispositivo cambia de estado.
- **xpl-stat** (estado): Este mensaje se envía para recuperar el estado de un dispositivo.

El mensaje está predefinido con una estructura fija como podemos ver en el siguiente ejemplo:

```
xpl-cmnd
{
    hop=1
    source=xpl-xplhal.myhouse
    target=acme-cm12.server
}
x10.basic
{
    command=dim
    device=a1
    level=75
}
```

Figura 3. Mensaje del tipo comando

Todos los mensajes están formados por las siguientes partes:

- Tipo de mensaje (xpl-cmnd, xpl-stat o xpl-trig)
- Cabecera del mensaje de tamaño variable (pero limitado). En esta parte deben definirse los datos siguiendo el patrón: nombre = valor.
- Esquema que resume la aplicación de la que procede el mensaje, de la forma: esquema. Tipo.
- Contenido del mensaje, definido también con el patrón: nombre = valor.

Capítulo 4. Herramientas

En este capítulo se presentan los diversos programas y dispositivos que hemos utilizado para la puesta en marcha de la parte práctica del proyecto.

4.1 Arduino

4.1.1 Introducción

Arduino es una herramienta electrónica de código abierto compuesta básicamente por una parte hardware que consiste en un microcontrolador y una parte software que permite la creación de los programas que serán almacenados en dicho hardware a través de un entorno específico creado para Arduino. Esta plataforma permite crear prototipos sencillos y fáciles de usar.

Por su propia definición, Arduino puede tomar datos reales del medio en el que se encuentra mediante sus pines de entrada y también puede controlar una gran variedad de actuadores como luces y motores entre muchos otros gracias a sus pines de salida. Este microcontrolador se programa mediante el lenguaje creado específicamente para Arduino (muy similar a C++) y el entorno de desarrollo Arduino. Este dispositivo permite la comunicación con los diferentes tipos de equipos y programas, de ahí el interés de utilizarlo en este proyecto.

Una de las ventajas de utilizar dicho microcontrolador es que existe un sitio web[1] dedicado donde se puede encontrar todo lo necesario para programar en Arduino y donde se puede descargar el entorno de desarrollo con las instrucciones de instalación y una guía de programas, ejemplos y librerías. En esta página encontramos una gran variedad de placas Arduino según sea el tamaño, el número de pines o el microcontrolador que lleve incorporado. También existen unas placas llamadas "*shield*" que permiten añadir funciones específicas al microcontrolador (*Ethernet*, WiFi , gestión de pantallas LCD, ...). Estas placas funcionan gracias a las librerías creadas para ese fin. En el Anexo I se describe detalladamente la placa Arduino.

4.1.2 Arduino *Ethernet Shield*

La placa Arduino que hemos utilizado en este proyecto es la Arduino *Ethernet*. Este tipo de placa permite conectar el microcontrolador a un puerto *Ethernet* lo que facilita el acceso y conexión a las diferentes redes que hemos implementado para nuestros tests.

[1] http://www.arduino.cc/

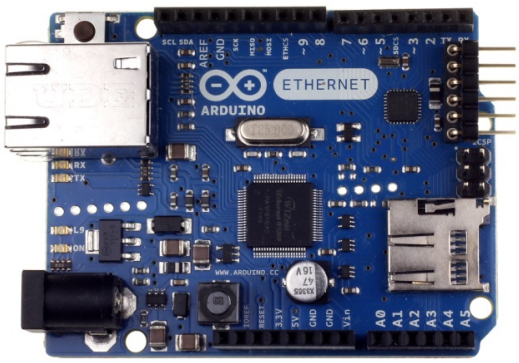

Figura 4. Vista frontal Arduino Ethernet

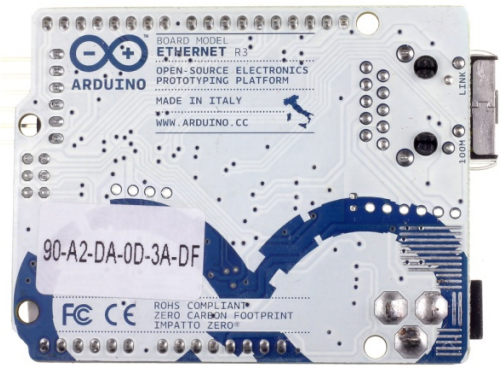

Figura 5. Vista posterior Arduino *Ethernet*

4.1.3 Tinkerkit

TinkerKit es un placa que superpuesta al microcontrolador Arduino permite conectar varios módulos y componentes a la misma. Se ha diseñado específicamente para la evaluación y creación rápida de prototipos y está equipada con múltiples conectores para la conexión de sensores, LEDs, interruptores ...
Esta placa ha permitido que hagamos los ejemplos más rápidamente gracias a las funciones que se incluyen en la biblioteca Tinkerkit.

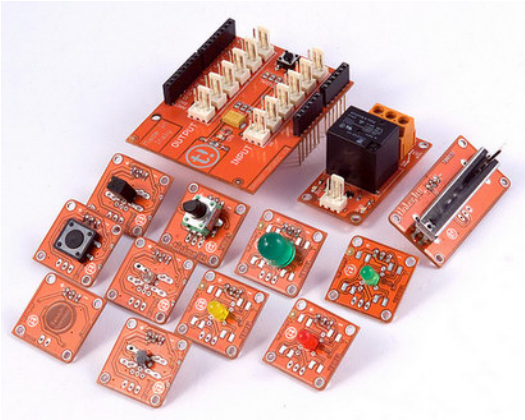

Figura 6. Placa y módulos Tinkerkit

4.2 LabVIEW

Después de interesarnos en el microcontrolador Arduino elegimos LabVIEW como herramienta para simplificar la creación de nuestros ejemplos.

LabVIEW (Laboratory Virtual Instrument Ingénierie Workbench), creado por National Instrument, es un entorno de programación gráfico utilizado comúnmente por ingenieros y científicos para desarrollar sistemas de medida, de tests y de control sofisticados, basados en el uso de iconos y conexiones virtuales que permiten una programación gráfica intuitiva.

LabVIEW se utiliza en aplicaciones que implican la adquisición, el control, el análisis y la presentación de datos. Además, LabVIEW permite la integración sencilla de instrumentos de medida y posee una gran variedad de bibliotecas integradas para el análisis avanzado y visualización de datos, lo que permite el desarrollo de nuevos instrumentos virtuales.

Los programas desarrollados en el entorno de programación de LabVIEW se llaman "instrumentos virtuales" (VIs), puesto que su apariencia y funcionamiento son similares a los de un instrumento de medida real. Los VIs están compuestos de una parte interactiva con el usuario y otra parte en la que se diseña el código. Al comparar la utilización con la de un instrumento

tradicional, un instrumento virtual proporciona más funcionalidad, alta conectividad y un amplio campo de utilización definido por el usuario.

El diagrama de LabVIEW está ligado a una interfaz gráfica de usuario llamada panel frontal. Cada VI está formado por tres componentes: un diagrama que integra el código gráfico, un panel frontal personalizable por el usuario y un panel de conexiones para los iconos que se utilizan como variables de entrada/salida.

4.3 Python

Python es una de las herramientas utilizadas durante la fase de desarrollo del proyecto. El interés en utilizar este lenguaje de programación aparece cuando implementamos uno de los experimentos con la placa Arduino. En este ejemplo, se pretende incluir un menú interactivo diseñado en Python para controlar el Arduino interactivamente.

Python es un lenguaje de programación orientado a objetos. Consiste en un lenguaje sencillo y con un enfoque de alto nivel. Podemos encontrar todo lo necesario para programar en Python en el Sitio Web Python[2] ya que es *open source*. En la misma página web existen enlaces a muchos módulos Python libres, programas, herramientas y documentación adicional.
El intérprete de Python es fácilmente extensible con nuevas funciones y tipos de datos implementados en C o C++ u otros lenguajes accesibles desde C como Arduino, de ahí el interés en su uso para este proyecto.

4.4 Wireshark

Una de las principales herramientas utilizadas ha sido Wireshark. Este programa es un analizador de protocolos *open source* que nos ha permitido analizar los paquetes existentes en nuestra red. Su característica principal es que es completamente transparente a la hora de analizar el tráfico de paquetes y permite la resolución de los posibles errores en la red.

El funcionamiento de Wireshark se basa en el uso de una amplia gama de filtros que permiten centrarse en el protocolo deseado en cada momento para el estudio de la red. Una vez seleccionado el paquete que va a ser analizado lo muestra desglosado en sus distintas partes para poder analizar también los distintos campos y cabeceras de cada una de las capas OSI.

La gestión de este programa nos ha permitido filtrar todos los paquetes que no tenían interés en este estudio concreto y ha facilitado enormemente el estudio del protocolo que nos interesaba.

[2] http://www.python.org/

4.5 Netdisturb

NetDisturb es programa de simulación de redes IP que puede generar las pérdidas que podemos encontrar en las redes IP. Algunos ejemplos son: latencia, retardo, *jitter*, limitación de ancho de banda, pérdida de datos, la duplicación y la modificación de paquetes ...

NetDisturb permite alterar los flujos en redes IP y favorecer así el estudio del comportamiento de aplicaciones, dispositivos o servicios en el entorno de una red "perturbada". El software se inserta entre dos segmentos *Ethernet* actuando como un puente y realiza la transferencia bidireccional de paquetes sobre las tarjetas de interfaz de red como *Ethernet*, *Fast Ethernet* y Gigabit.

Para conectar las dos redes a dicho software basta con conectar la primera red a la tarjeta *Ethernet* que se encuentra en el equipo en el cual se ejecuta NetDisturb y hacer lo mismo con la segunda red y la otra tarjeta *Ethernet* de la que dispone el PC.

Este software nos permitió simular el enlace satelital con un retardo de 600 milisegundos debido a la ida y la vuelta de la información. El retardo de transmisión (latencia) entre el satélite y la antena terrestre corresponde principalmente a la transferencia de información por los 36 000 a 41 000 kilómetros que separan el satélite de la superficie de la tierra. Por supuesto hay que añadir el retraso debido a la congestión que se puede dar a nivel de los receptores, el tiempo de paso por los amplificadores, etc. Para una simple transferencia de la información en un solo sentido el retardo sería de unos 120ms al ecuador y de unos 140 ms al polo. Por tanto si se requiere una ida y vuelta es necesario duplicar este tiempo. Los enlaces de internet vía satélite están todavía más penalizados porque los protocolos que se utilizan suelen requerir de envíos que recorren los enlaces en los dos sentidos para confirmar la recepción de los datos (es lo único que nos garantiza que la información enviada se ha recibido). Es por eso que se llega tan rápido a retardos de hasta medio segundo.

La siguiente figura muestra el diagrama de funcionamiento de Netdisturb:

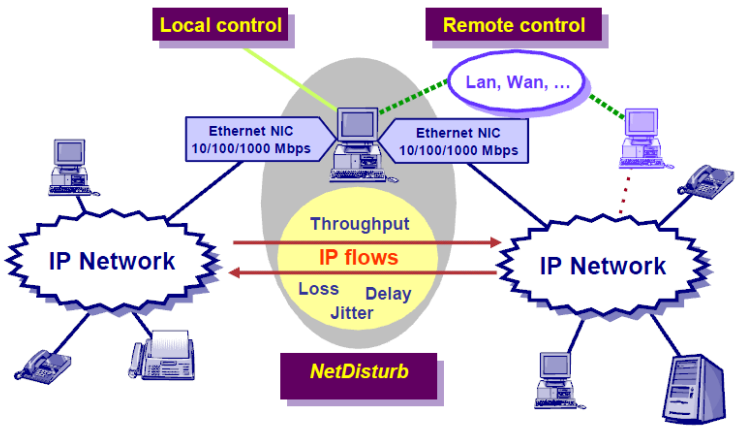

Figura 7. Esquema de funcionamiento de Netdisturb

Capítulo 5. Trabajo experimental y resultados

Después de haber dedicado la primera parte del proyecto a la documentación sobre la domótica, a los protocolos existentes en el mercado y a la reflexión sobre la forma y los medios para poner en práctica la realización del proyecto, en la segunda parte del proyecto se trabajó en la puesta en marcha de las arquitecturas y en la realización de las distintas pruebas en las que dividimos el proyecto para llegar al escenario final donde se integran todos los subsistemas que fueron llevados a cabo.

Al no disponer de equipos domóticos reales para nuestras pruebas era necesario encontrar dispositivos que nos permitiesen realizar estas pruebas de manera poco costosa. El sistema Arduino parecía cumplir nuestras expectativas porque era económicamente accesible y transparente, es decir, que el usuario no necesita conocer el funcionamiento interno del microcontrolador Arduino para poder utilizarlo. Arduino hace que sea posible la conexión de dispositivos de diferentes marcas lo que hace la integración de todos los dispositivos mucho más fácil. Existe mucho código libre disponible y es fácil de programar.

Dado que el objetivo final es establecer una comunicación vía satélite utilizando un transmisor y un receptor de manera que podamos verificar los requisitos necesarios para la puesta en marcha del enlace satelital, fuimos realizando varias experiencias que poco a poco nos permitieron establecer dicho enlace.

Presentamos a continuación los subsistemas que fuimos definiendo explicando los montajes utilizados, la problemática relacionada con la puesta en marcha de las mismas y los resultados obtenidos.

5.1 Subsistema 0. LabVIEW

Para simplificar la programación con Arduino existe una herramienta llamada LIFA (LabVIEW Interface For Arduino). Se trata de una extensión de LabVIEW (a partir de la versión 2009) que permite controlar una placa compatible Arduino desde LabVIEW.

El primer ejemplo realizado durante el desarrollo de la parte práctica se realizó con LabVIEW 2012 y la extensión LIFA. La ventaja de este programa es que la interfaz gráfica llamada "panel frontal" se crea a medida que se diseña gráficamente el diagrama seleccionando e uniendo los diferentes bloques.

Este ejemplo consiste en leer un archivo de texto y ejecutar la acción correspondiente según la palabra contenida en dicho archivo. Elegimos este ejemplo con el fin de reconocer palabras clave que nos permitan desencadenar determinadas acciones y así poder gestionar una casa con un sistema de reconocimiento vocal que convierta la voz en texto.

5.1.1 Diagrama de bloques

El siguiente diagrama es el código gráfico implementado en LabVIEW con el que conseguimos abrir el archivo en cuestión, leer su contenido y ejecutar la acción correspondiente que en nuestro caso consistía en encender o apagar el led verde situado en la parte derecha del diagrama. La programación resulta bastante sencilla gracias a las funciones prediseñadas existentes en LabVIEW y que corresponden con los bloques amarillos que vemos en el diagrama.

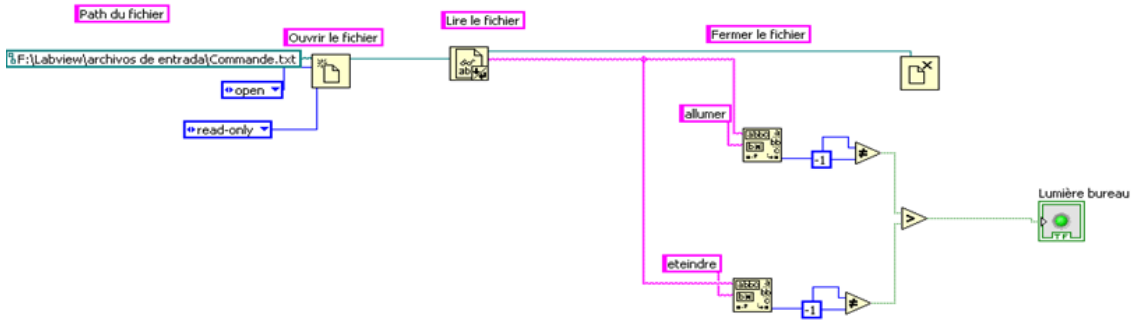

Figura 8. Diagrama gráfico LabVIEW

5.1.2 Panel frontal

A continuación se presenta el panel frontal que se crea simultáneamente mientras se diseña el diagrama de bloques y que muestra el resultado visual del código implementado.

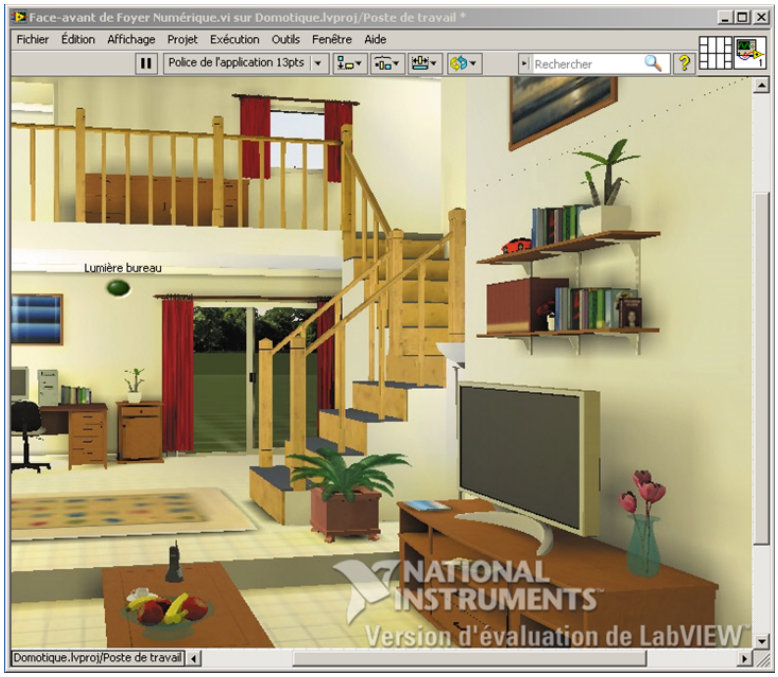

Figura 9. Panel frontal asociado al diagrama

La ventaja de trabajar con LabVIEW es que podemos, por ejemplo, añadir un fondo de pantalla que puede hacer que la interfaz creada sea más realista.

El resultado mostraba que el LED se iluminaba si en el archivo aparecía la palabra clave "*on*" o se apagaba si la palabra era "*off*". Modificando ligeramente el código podríamos no desencadenar ninguna acción si la palabra contenida en el fichero no corresponde con ninguna de las palabras clave.

5.2 Subsistema 1. Pulsador - Led

Para aprender a manejar las placas Arduino y familiarizarse con el lenguaje de programación correspondiente adaptamos el siguiente ejemplo.

El propósito de este ejemplo era comunicar dos placas Arduino entre ellas y ejecutar comandos que pueden ser útiles para administrar un hogar a distancia como encender la luz del salón (u ordenar cualquier otra acción) presionando por ejemplo un botón (en nuestro caso concreto, el pulsador Tinkerkit que aparece en la siguiente imagen) en un panel de control remoto al dispositivo que se desea controlar.

La definición de una placa como emisora y la otra como receptora nos permitió comenzar a establecer una comunicación sencilla entre las dos placas, para posteriormente ir incrementando la dificultad, añadiendo otras etapas para llegar a comunicarlas por medio de un satélite. De ahí la importancia de este ejemplo del que presentamos el esquema a continuación.

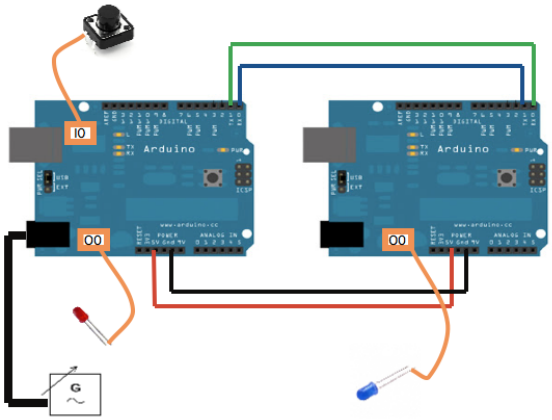

Figura 10. Esquema Pulsador-Led

5.2.1 Funcionamiento

El funcionamiento del programa se basa en la transmisión a través del puerto serie de ciertos caracteres (en este caso los caracteres "a" y "b") que son enviados por la placa emisora cuando presionamos el pulsador. La placa receptora es la responsable de leer los datos entrantes por el puerto serie y en función del valor de la lectura del carácter, encender o apagar el led que está conectado a dicha placa.

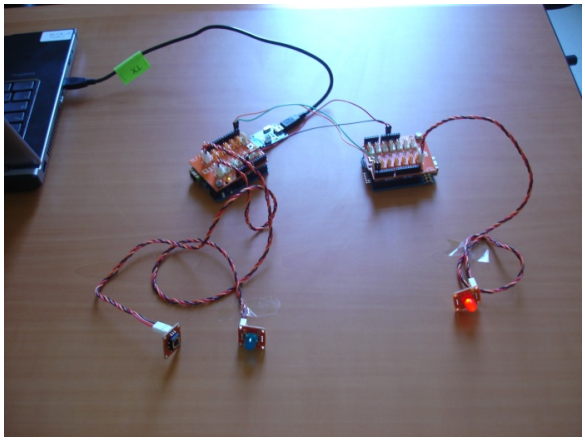

Figura 11. Situación inicial

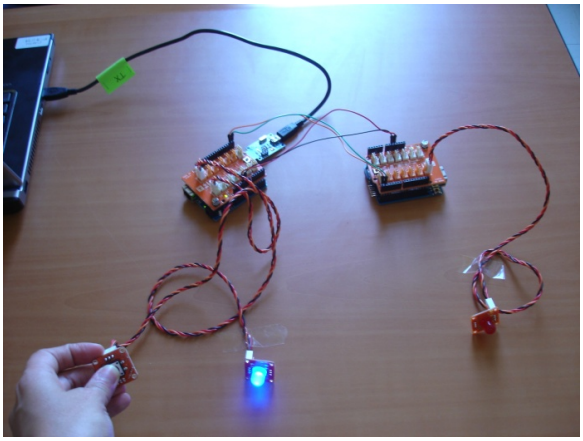

Figura 12. Pulsado del interruptor

Las imágenes siguientes muestran cómo se apagaba el led rojo una vez que presionábamos el pulsador y a su vez se encendía el led azul, lo que indica el correcto envío del carácter “a” y su correcta recepción. El código correspondiente a este escenario lo encontramos en el Anexo II.

El interés de intercambiar texto en lugar de señales es la posibilidad de utilizar un sistema de reconocimiento vocal como hemos mencionado en el escenario anterior. Con este sistema podríamos controlar la vivienda inteligente por medio de comandos de voz como "encender la luz", "cerrar las persianas" ... El sistema transformaría los comandos de voz en texto y el envío del texto en este ejemplo daría lugar a acciones de manera similar.

5.3 Subsistema 2. Arduino y *Ethernet*

Los siguientes ejemplos permiten el uso de *Ethernet* con las placas Arduino. En nuestro caso nos será muy útil ya que la idea es crear dos redes independientes y posteriormente establecer la comunicación entre ellas vía satélite. Es por esta razón que, si es posible el uso de *Ethernet*, la integración de Arduino en las redes será mucho más fácil ya que el "box" proporcionado por los proveedores de acceso a Internet (terrestre o satélite) ofrecen todas conectividad *Ethernet*.

5.3.1 *Ethernet* 1. Construcción de una red local

En este primer ejemplo se define una red local formada por una placa Arduino y un PC, con el propósito de establecer el microcontrolador Arduino como servidor http gracias a la conexión *Ethernet*. Este ejemplo nos ha permitido crear una red LAN inicial para posteriormente poder añadir más componentes a la red.

5.3.2 *Ethernet* 2. Cliente - Servidor

Ampliando el código anterior conseguimos establecer la tarjeta Arduino como servidor y el navegador Firefox como cliente. El programa que corre en el microcontrolador Arduino envía un código hacia el navegador y este a su vez lo muestra por pantalla en forma de texto simple. Dicho proceso se puede controlar simultáneamente gracias al monitor serie del entorno Arduino.

Gracias a estos ejemplos hemos conseguido el siguiente paso que es el uso del microcontrolador como un dispositivo más de una red LAN.

Los códigos correspondientes se encuentran en el Anexo III junto con las diferentes especificaciones necesarias para establecer dichas redes y las imágenes que muestran el correcto funcionamiento.

5.4 Subsistema 3. Interactuar gracias a Python

La domótica ofrece la posibilidad de comunicarse con los diferentes dispositivos de la casa de forma interactiva. Podemos hacerlo a través de un menú, en una pantalla colocada a la entrada de la casa, desde un ordenador, un teléfono móvil, una tablet ...

Es por eso que decidimos llevar a cabo este ejemplo, con el propósito de establecer el Arduino como servidor y un ordenador como cliente que nos permita utilizar un menú interactivo creado con el lenguaje de programación Python. La explicación detallada se encuentra en el Anexo IV.

Con este ejemplo mostramos que se pueden activar acciones de manera remota a través de otros dispositivos colocando placas Arduino en las distintas estancias de la casa. Este paso también nos permite acercarnos a la gestión de la casa sin pensar en la distancia a la que nos encontramos de la vivienda en cuestión.

5.5 Subsistema 4. Creación de una red LAN añadiendo un segundo servidor

Se trata de un ejemplo próximo al anterior con el que se pretende añadir otra tarjeta Arduino a la red. El interés de añadir otro periférico es la necesidad de instalar un *switch* en dicha red para poder interconectar todos los dispositivos.

El objetivo de esta prueba es acercarnos cada vez más a la simulación de una vivienda real pudiendo elegir entre las diferentes estancias de la casa, es decir, en nuestro caso particular, entre las diferentes placas Arduino colocadas en la red. La explicación detallada se encuentra en el Anexo V.

5.6 Subsistema 5. Uso de una tarjeta de memoria microSD

Puesto que las placas Arduino tienen una memoria interna reducida y nosotros necesitamos más espacio para almacenar código o crear y leer ficheros, existe la posibilidad de ampliar dicha memoria con una microSD.

En nuestro caso, es una opción bastante interesante puesto que es de gran interés poder leer y crear ficheros para poder registrar más información como el tiempo, la temperatura... El código con el que ponemos a punto la placa para permitir el uso de la microSD se encuentra en el Anexo VI.

5.7 Sistema domótico. Protocolo domótico xPL

La siguiente etapa consiste en introducir el protocolo domótico xPL para conectar los diferentes dispositivos de la casa entre sí. Como ya hemos explicado con anterioridad, el protocolo xPL (extremely simple protocol) dispone de funciones de auto-descubrimiento y auto-configuración que le permiten ser *« Plug and Play »* contrariamente a muchos de los otros protocolos domóticos.

El objetivo de llevar a cabo este escenario es poner en marcha el protocolo elegido en este proyecto y verificar su funcionamiento para poder alcanzar uno de los objetivos fundamentales del proyecto. El principio consiste en lo siguiente, cada máquina (la mayor parte del tiempo un PC pero no necesariamente tiene por qué serlo) que forma parte de una red xPL tiene instalada un "*hub* xPL" que permite encaminar los mensajes entre los diferentes clientes xPL que existen en dicha máquina, pero además debe difundir estos mensajes a la red local a la que pertenece. Estos mensajes serán recibidos a su vez por los otros "*hubs* xPL" y en consecuencia por los clientes conectados a estos "*hubs*".

En este ejemplo hemos ido introduciendo progresivamente las diferentes herramientas de las que disponíamos permitiendo así el correcto funcionamiento del protocolo xPL en la red. Para ello nos servimos de la biblioteca xPL disponible en la página web[3].

5.7.1 Envío de paquetes "*heartbeat*"

Para este ejemplo partimos de una red LAN formada por un solo PC y por un microcontrolador Arduino. El rol del PC es el de alojar el "*hub* xPL" que permitirá enviar los mensajes a los diferentes dispositivos xPL que dicho PC gestiona (en esta primera etapa sólo el Arduino). Estos mensajes son los que nos permiten conocer los dispositivos presentes en la red a medida que van conectándose. El rol de la placa Arduino es ser vista como un periférico que trabaja con el protocolo xPL, enviando periódicamente mensajes (*heartbeat*) a la red para informar de su presencia en la misma. Dichos mensajes son

[3] http://connectingstuff.net/blog/connectingstuff/

enviados a todos los equipos que tengan asignada una red ip, pero, dado que no es posible escuchar varias aplicaciones al mismo tiempo en un sólo puerto, el "*hub* xPL" que corre en el PC reenvía los mensajes a todas las aplicaciones que están conectadas a dicho PC. Gracias a todo este proceso se construye una lista de las aplicaciones existentes, de ahí el interés del envío de los llamados *hearbeats*.

En dicho ejemplo y tal y como se detalla en el Anexo VII, vemos como efectivamente el microcontrolador Arduino envía cada 10 segundos un paquete *heartbeat* para indicar su presencia en la red. Como podemos comprobar en la captura de pantalla (Figura 30) del Anexo VII, la dirección de envío de estos paquetes es la dirección *broadcast* correspondiente a la red en la que se encuentra. Es importante recordar que no se trata del broadcast conocido comúnmente sino que es un *broadcast* específico del protocolo, en el sentido que es enviado a todos los elementos de la red xPL.

5.7.2 Establecer una red xPL con las placas Arduino

Para crear una red xPL se debe tener en cuenta que el protocolo presenta una serie de restricciones. Una de las restricciones es que en un PC sólo puede haber una aplicación que reciba mensajes por el puerto designado (el protocolo xPL tiene asignado el puerto 3865). Para salvar esta restricción es necesario que cada PC que albergue al menos una aplicación xPL ejecute a su vez "*hub* xPL". Dicho *hub* estará a las escucha de los mensajes xPL que provengan del puerto 3865 y cada vez que reciban un mensaje lo retransmitirá al resto de los programas que se estén ejecutando en el PC. El *hub* es conocedor de los programas que están siendo ejecutados y de los puertos que están a la escucha gracias a la información proporcionada por el envío periódico de los *heartbeat* en la red. De esta forma eliminamos la problemática que impone el protocolo al trabajar sólo con el puerto 3865.
El envío de mensajes en una red xPL es mucho más simple que la recepción de los mismos. Las aplicaciones xPL sólo se dedican a enviar estos mensajes de tipo *broadcast* a la red a la que pertenecen. Son los *hubs* los que manejan verdaderamente la comunicación entre los Pcs, puesto que reciben dichos mensajes y los retransmiten a las aplicaciones xPL locales. Por tanto el componente principal de una red xPL es el *hub*. En nuestro caso particular hemos usado el *hub* propuesto en la página web[4]. Se trata de un *hub* que funciona en *Java* y lo elegimos puesto que este *hub*, llamado xPL4Java, es una plataforma que puede ejecutarse en cualquier ordenador que disponga de *Java*. En el Anexo VII se detalla el proceso seguido y los códigos utilizados.

Con este ejemplo llegamos a comprobar que efectivamente a partir del momento en el que conectamos el segundo Arduino, empiezan a aparecer en la ventana de Wireshark los paquetes *heartbeat* con la dirección ip asignada a dicho periférico.

[4] http://www.xpl4java.org/

5.7.3 Seguimiento de la red xPL → xPLHal

xPLHal es el corazón de un sistema domótico xPL instalado en Windows. Es el responsable del tratamiento de los mensajes xPL entrantes, del envío de nuevos mensajes, de la ejecución de los programas y permite la configuración de periféricos xPL. La red final implementada para verificar el funcionamiento del protocolo xPL es la siguiente:

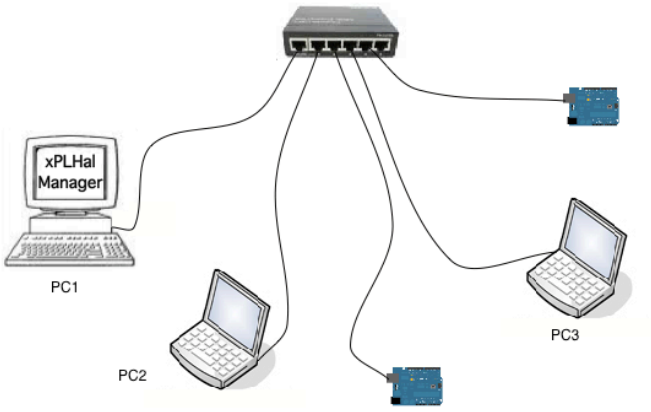

Figura 13. Red xPL final

El programa xPLDiag, encargado del chequeo de los diferentes componentes de la red xPL para su correcta puesta en marcha, se instaló en uno de los ordenadores que forma parte de la red. Dicho programa nos permite visualizar todas las aplicaciones existentes en ese momento en la red. Las imágenes de dichas aplicaciones se adjuntan en el Anexo VII (Figura 31). Una vez visualizada la correcta inicialización de las aplicaciones, comprobamos que el envío de paquetes era viable definiendo los paquetes en el programa xPLHal Manager (envío detallado en el Anexo VII, (Figura 32)) y observamos en el monitor xPL que los mensajes habían sido enviados como se muestra en la siguiente figura, donde el paquete seleccionado en dicho monitor es el que se ha enviado a través de dicho programa.

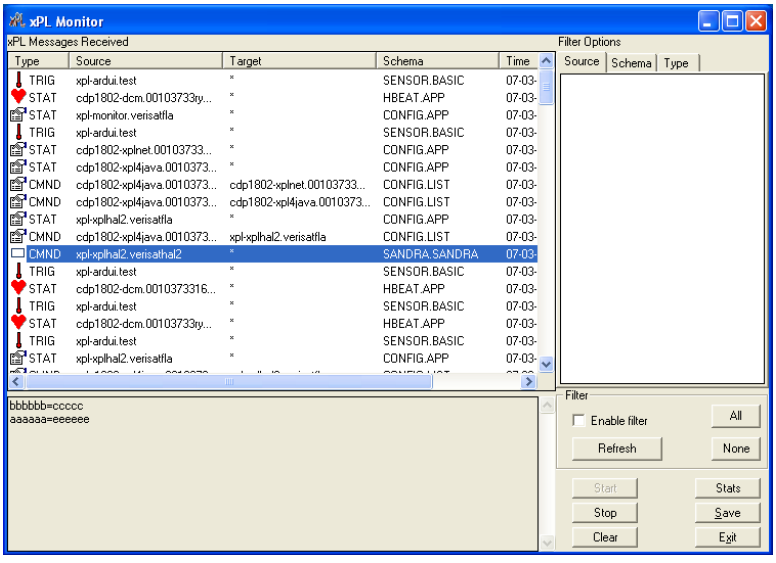

Figura 14. Monitor xPL

La representación visual de los diferentes tipos de paquetes con distintos iconos, como se ve en la ilustración precedente, hace que el análisis de los paquetes se realice de una forma mucho más rápida y sencilla. En el monitor se observan varios *heartbeat* (simbolizados en el programa con el corazón rojo) provenientes de las diferentes aplicaciones existentes en la red, mensajes de *trigger*, de comando y de estado, enviados por las máquinas y que sirven para configurar la red.

Para verificar el correcto funcionamiento procedimos al envío de un paquete desde el PC1 donde se encuentra instalado dicho programa y analizamos con Wireshark el proceso seguido por dicho paquete. Efectivamente vimos como el paquete había sido enviado, pero vimos también que dicho paquete fue reenviado por otro de los Pcs, el PC2, hacia el resto de los dispositivos existentes en la red. Hecho que muestra el correcto funcionamiento del protocolo xPL ya que es el *hub* instalado en este segundo PC el que reenvía el paquete redireccionándolo hacia todas las máquinas de la red. La captura de pantalla que lo muestra se encuentra en el Anexo VII (Figura 33).

Con este ejemplo queda demostrado que el protocolo funciona como se define en la teoría, cubriendo así uno de los objetivos propuestos, es decir, hemos verificado la existencia de las funciones de auto-descubrimiento y auto-configuración que permiten al protocolo xPL ser *« Plug and Play »*. Gracias a Wireshark hemos visto pasar todos los paquetes emitidos por cada aplicación y hemos visto también como eran difundidos por los *hubs* hacia todas las aplicaciones asociadas a esos *hubs*, creando así la lista de dispositivos existentes.

Para concluir, el protocolo xPL presenta ciertas ventajas como la simplicidad de conexión de equipos de distintas marcas. Otra ventaja es la capacidad del protocolo al ofrecer la conexión *« Plug and Play »*. Esta propiedad permite añadir sin dificultad nuevas aplicaciones y nuevos periféricos xPL en la red. Las diferentes herramientas xPL existentes ayudan a monitorizar los intercambios que se producen en la red, a gestionar el envío de mensajes y permite la configuración de los dispositivos xPL en la red.

Sin embargo, uno de los mayores inconvenientes es el envío de mensajes de tipo *broadcast* que permiten a los *hubs* descubrir el estado de la red en cada instante, pero que no pasan por los *routers* ya que estos no dejan pasar por definición los mensajes del tipo x.x.x.255. El protocolo trabaja a nivel aplicación por lo que una posible solución al problema sería la utilización de pasarelas.

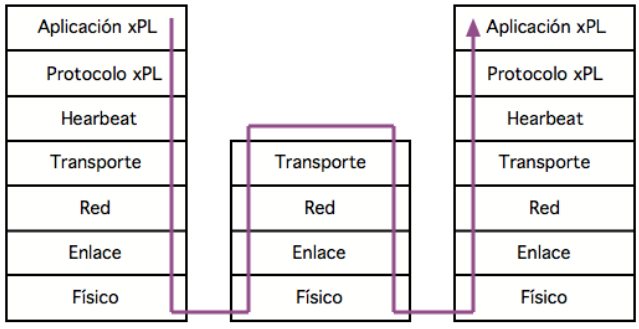

Figura 15. Esquema pila OSI

Para ello, habría que colocar la pasarela en el enlace de subida y en el de bajada. El rol de la pasarela del enlace descendente será desencapsular la cabecera añadida por la pasarela del enlace ascendente para poder encaminar los paquetes hacia todos los periféricos.

Otra solución posible consistiría en subir los paquetes xPL en un servidor Web accesible por todos los periféricos xPL. Los clientes podrían interrogar el servidor periódicamente y activar las órdenes pertinentes.

Otro inconveniente es que la puesta en marcha del protocolo con los programas que hemos utilizado no es evidente. La inicialización de la red xPL es caprichosa y requiere bastante tiempo. La definición y puesta a punto de la red xPL requiere un procedimiento particular detallado en el Anexo VII (inicialización de los *hubs*, lanzamiento del programa xPLHal Manager, lanzamiento del programa xPLDiag…) que reduce en gran medida estas dificultades.

5.8 Escenario final. Envío de la información vía satélite

La última etapa de la parte experimental del proyecto se basa en el envío de los datos vía satélite para gestionar la vivienda a distancia y en el estudio de la viabilidad del enlace propuesto.
El esquema propuesto y llevado a cabo para comunicar dos redes independientes a través del satélite se presenta en el siguiente esquema:

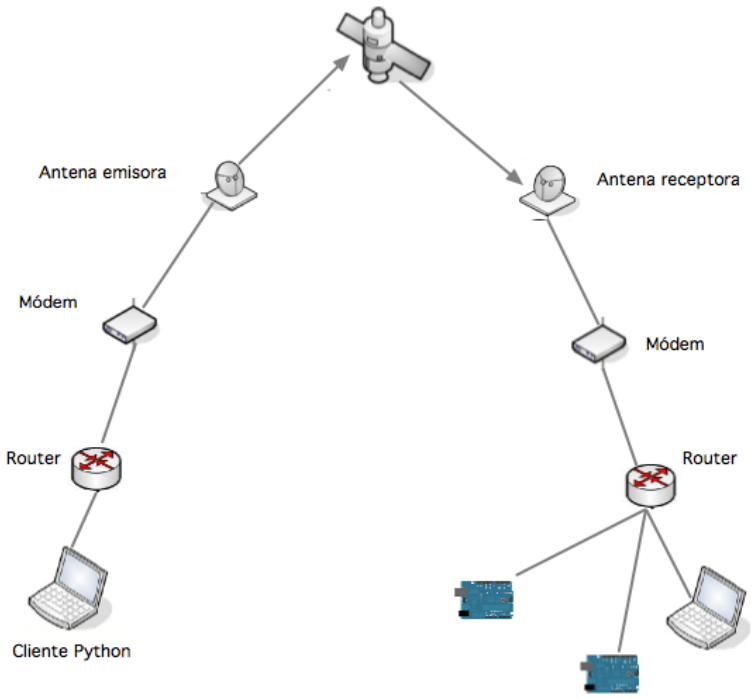

Figura 16. Esquema satelital

El problema impuesto por el protocolo xPL para poder llevar a cabo esta etapa está ligado a las características fundamentales en las que se basa dicho protocolo. El hecho de que el protocolo utilice periódicamente el *broadcasting* para conocer los dispositivos que se encuentran conectados a la red en cada momento, impide el envío de datos vía satélite puesto que la presencia de los *routers* para establecer el enlace no permiten el envío de paquetes mediante *broadcast*. Desafortunadamente, la proximidad al final de mi estancia en la empresa y el retraso en cuanto a la recepción del material necesario para poner en práctica esta última etapa en el laboratorio, no nos permitió solucionar este problema pero se procedió al estudio de las posibles soluciones al mismo.

Para analizar el comportamiento del enlace satelital retomamos el subsistema 4, en el cual manejábamos las placas Arduino gracias al menú implementado con Python, el cual nos permitía seleccionar la placa con la que queríamos interactuar.

Antes de comunicarnos a través del satélite real, realizamos una simulación con el programa Netdisturb con el fin de observar el comportamiento del protocolo xPL y el efecto de los retrasos debidos al satélite. Así pues, conectamos el PC en el que se ejecutaba el menú Python a una de las tarjetas *Ethernet* del PC que albergaba Netdisturb para simular el cliente y las placas Arduino a la segunda tarjeta *Ethernet* de dicho PC para simular los periféricos xPL que representan distintas estancias de la casa.

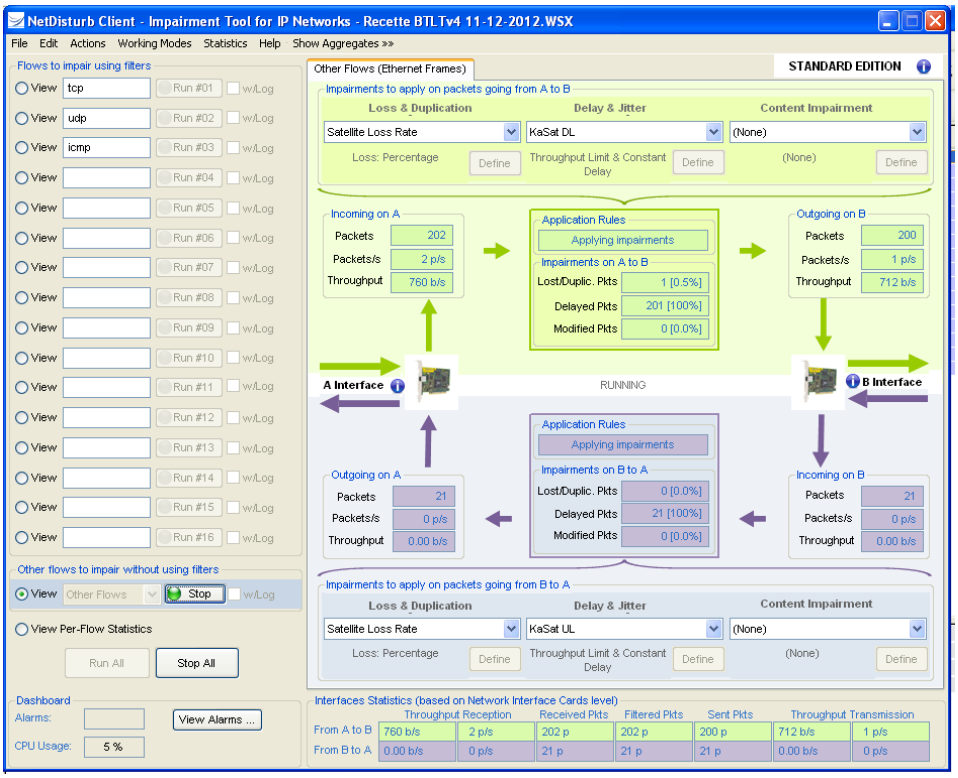

Figura 17. Simulación del envío por satélite con NetDisturb

La captura precedente muestra la simulación realizada en la cual, hemos introducido un retardo de 300ms debidos al tiempo de transferencia de los datos a través del satélite y a la transferencia por los equipos existentes en la red tanto en el enlace ascendente como en el descendente. Para probarlo enviamos paquetes ICMP a las placas Arduino. Como vemos en la siguiente captura de la consola cmd, los paquetes enviados presentan un retardo medio de 699 ms correspondientes a la suma de los retardos de los enlaces de subida y bajada.

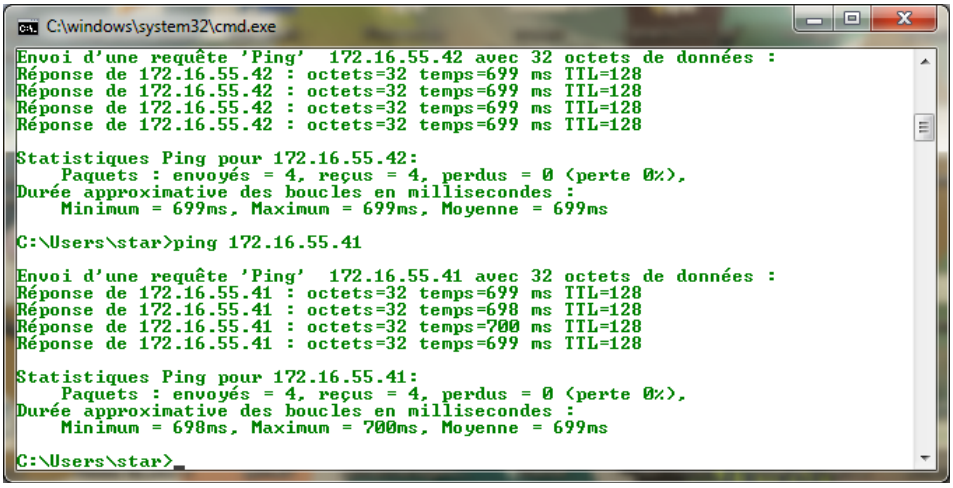

Figura 18. Ping a las placas Arduino para verificar el retraso introducido con NetDisturb

Una vez realizada la simulación introdujimos el satélite real en la red. El primer problema que encontramos era debido al envío de paquetes puesto que no conocíamos la dirección pública del *router* que estaba en el enlace descendente y en consecuencia no podíamos encaminar los paquetes enviados. La necesidad de conocer la dirección pública por la que los paquetes entran al *router* descendente, para seguidamente ser encaminados hacia los Arduino correspondientes, se debe a la definición del menú interactivo creado con Python. En dicho código es necesario poner las direcciones destino de cada una de las instrucciones, por tanto, si estas se desconocen, las instrucciones no llegarían nunca al destino deseado.

Una de las soluciones propuestas para solucionar este problema era averiguar las direcciones públicas de los *routers* a partir de alguna de las aplicaciones existentes en la web. En nuestro caso pudimos seleccionar esta opción porque teníamos acceso a los dos *router* de los dos enlaces.

La asignación de las direcciones de los equipos conectados a los *routers* se hacen en DHCP por lo que son desconocidas para nosotros. Para conocerlas nos servimos de la función ping. Una vez conocidas, configuramos los *routers* mediante « *port forwarding* » de manera que los paquetes entrantes con un determinado número de puerto sean encaminados hacia las placas Arduino con el mismo número de puerto asignado. Para ello utilizamos una asignación de la forma:

@publica puerto1 → @Arduino 1 puerto1
@publica puerto2 → @Arduino 2 puerto2

Esta asignación la realizamos gracias al programa dd-wrt asociado a los *routers* Linksys que han sido utilizados en nuestros enlaces. Podemos observar dicha asignación en la imagen siguiente:

Figura 19. Port forwarding de las placas Arduino

Una vez establecida la conexión, procedimos al envío de comandos a las placas Arduino sin obtener el resultado esperado. Puesto que los *routers* daban problemas, decidimos conectar las redes directamente a los módems para evitar el paso por los *routers*. Pero el problema persistía, por lo que intentamos buscar el origen del problema. Después de diversas pruebas, comprobamos que conseguíamos hacer ping con el ordenador conectado en la red situada al otro lado del satélite pero no a las tarjetas Arduino conectadas a la misma red. Esto tiene su lógica, ya que el hecho de conectar las redes directamente a los módem impone la existencia de una sola dirección al otro lado del módem. Dicha dirección ip es la dirección que toma el PC que está conectado en esta red, por tanto, las tarjetas Arduino se convierten en terminales y es por este motivo que no responden a los ping, ya que carecen de dirección ip. En este caso, una vez más, la solución propuesta sería el uso de pasarelas, pero la finalización de mi estancia en la empresa no nos permitió seguir con las pruebas.

Capítulo 6. Conclusiones

El resultado de la elaboración de este proyecto ha sido satisfactorio puesto que se han cumplido en su mayor parte los objetivos propuestos y se han ofrecido posibles alternativas a los inconvenientes que se han encontrado.

Se trata de un buen protocolo en cuanto a su simplicidad a la hora de interconectar equipos de distintas marcas, lo que nos permitió poder añadir fácilmente nuevas aplicaciones y periféricos xPL a nuestra red. Esta funcionalidad es de gran interés, puesto que la probabilidad de añadir nuevos equipos a una red doméstica es muy alta. Sin embargo, impone bastantes problemas el hecho de que trabaje con el envío de los mensajes mediante *broadcast*, ya que hace imposible el uso de *routers* para la creación de la red. Las soluciones propuestas a dicho problema serían el uso de pasarelas o el uso de un servidor Web para el almacenamiento de la información utilizada en la red. Por otro lado, como comprobamos durante mi estancia en la empresa, se trata de un protocolo que requiere una inicialización demasiado caprichosa y necesita de bastante tiempo para ponerse en marcha.

La puesta en marcha del escenario final se vio afectada en su mayor parte por el retraso en la recepción de los componentes necesarios y la puesta a punto del laboratorio en el que se llevó a cabo la transmisión vía satélite. Sólo se pudieron estudiar los primeros inconvenientes a la hora de llevar a cabo el envío de la información. De todas formas, el estudio de dichos inconvenientes formaba parte de los objetivos de este proyecto y ya partíamos con el inconveniente añadido debido a la ausencia de documentación sobre la domótica vía satélite. La conclusión a la que se llegó y por la que se retomará este proyecto, será el uso de pasarelas para el establecimiento de la red, evitando así la inundación de la red debido al envío de los paquetes de tipo *broadcast*.

La realización de este proyecto me ha permitido adquirir numerosos conocimientos ligados a un dominio tan amplio y diverso como la domótica, las redes, la informática, la electrónica, las radiofrecuencias, …

He entendido la importancia de tener una visión general del proyecto como un todo y de implementarlo en varias etapas, aumentando progresivamente la dificultad para llegar a conseguir los objetivos propuestos.

He aprendido también a trabajar con perseverancia aunque las circunstancias no fueran las mejores, ya que hemos encontrado un cierto número de problemas técnicos con los equipos utilizados para hacer nuestros tests (problemas de prestaciones de los ordenadores, de configuración de los firewalls, de compatibilidad entre aplicaciones y el sistema de explotación,…)

Por otro lado, he encontrado muy enriquecedor el hecho de realizar el proyecto en el organismo CNES, dónde se me ha permitido descubrir el mundo laboral en un centro de investigación de tal envergadura. Trabajar con profesionales como los compañeros que tuve en mi departamento solo me ha dejado muy buenas experiencias que sin duda volvería a repetir.

Bibliografía

1. **Khusvinder Gill, Shuang-Hua Yang, Fang Yao, and Xin Lu**. *A ZigBee-Based Home Automation System*. IEEE Transactions on Consumer Electronics, Vol. 55, No. 2, MAY 2009.
2. **Miguel A. Zamora-Izquierdo, José Santa, Antonio F. Gómez-Skarmeta**. *An Integral and Networked Home Automation Solution for Indoor Ambient Intelligence.*
3. **Caio Augustus Morais Bolzani, Cristian Montagnoli, Marcio Lobo Netto.** *Domotics Over IEEE 802.15.4 – A Spread Spectrum Home Automation Application*.
4. **Somak R. Das, Silvia Chita, Nina Peterson, Behrooz A. Shirazi, and Medha Bhadkamkar**. *Home Automation and Security for Mobile Devices*.
5. **A.J. Bernheim Brush, Bongshin Lee, Ratul Mahajan, Sharad Agarwal, Stefan Saroiu, Colin Dixon**. *Home Automation in the Wild: Challenges and Opportunities.*
6. **Dr Leslie Haddon.** *Home Automation: Research issues. Paper presented at the second EMTEL Workshop: 'The European Telecom User'. November 10-11th, 1995, Amsterdam, The Netherlands*.
7. **Carelin Felix and I. Jacob Raglend**. *Home Automation Using GSM.* Proceedings of 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies.
8. Centro Nacional de estudios espaciales. http://www.cnes.fr [En línea]
9. Centro espacial de Toulouse. http://www.capcomespace.net/dossiers/espace_europeen/CST/CST.htm [En línea]
10. Domótica. http://doc.ubuntu-fr.org/domotique [En línea]
11. Domótica. http://domotique.comprendrechoisir.com [En línea]
12. Protocolo X10. http://www-igm.univ-mlv.fr/~dr/XPOSE2007/aessaidi-ndiop_LA-DOMOTIQUE/infraClassique.htm#X10 [En línea]
13. Protocolo KNX. http://domotique.comprendrechoisir.com/astuce/voir/155606/le-protocole-knx-comment-ca-marche-pour-qui [En línea]
14. Protocolo EHS. http://www.jaec.info/Home%20Automation/Protocols-buses-house/Ehs-Protocol/ehs-protocol.php [En línea]
15. Protocolo X2D. http://domotique.comprendrechoisir.com/astuce/voir/155607/le-protocole-x2d-x3d-comment-ca-marche-pour-qui [En línea]
16. Protocolo Z-Wave. http://maison-et-domotique.com/2012/03/14/quest-ce-que-le-z-wave/ [En línea]
17. Protocolo OpenWebNet. http://fr.wikipedia.org/wiki/Openwebnet [En línea]
18. Protocolo xPL. http://www.planete-domotique.com/blog/2011/05/18/le-protocole-xpl/

19. Protocolo xPL.
http://xplproject.org.uk/wiki/index.php?title=Main_Page [En línea]
20. Protocolo xPL
http://en.wikipedia.org/wiki/XPL_Protocol [En línea]
21. Protocolo xPL.
http://www.poulpy.com/2010/02/xpl-one-protocol-to-rule-them-all/ [En línea]
22. Protocolo XAP.
http://www.xapautomation.org [En línea]
23. Proyecto xPL.
http://xplproject.org.uk/wiki/index.php?title=XPL_Specification_Document [En línea]
24. Internet vía satélite.
http://fr.wikipedia.org/wiki/Internet_par_satellite [En línea]
25. Labview.
http://labview.developpez.com/faq/?page=2-1 [En línea]
26. Placa Arduino.
http://www.lextronic.fr/P23528-platine-sensor-shield-pour-arduino.html [En línea]
27. Programación sockets.
http://venom630.free.fr/geo/tutz/programmation/python/sockets/ [En línea]

Anexos

Anexo I. Descripción detallada Arduino

Descripción

La Arduino *Ethernet* es una placa electrónica basada en el microprocesador Atmega328. Cuenta con 14 pines de entrada/salida digitales, 6 entradas analógicas, un oscilador de cuarzo de 16 MHz, una conexión RJ45, una toma de alimentación y un botón de reinicio.

Tabla de características

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage Plug (recommended)	7-12V
Input Voltage Plug (limits)	6-20V
Input Voltage PoE (limits)	36-57V
Digital I/O Pins	14 (of which 4 provide PWM output)
Arduino Pins reserved:	10 to 13 used for SPI
	4 used for SD card
	2 W5100 interrupt (when bridged)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
W5100 TCP/IP Embedded Ethernet Controller	
Power Over Ethernet ready Magnetic Jack	
Micro SD card, with active voltage translators	

Tabla 1. Características Arduino *Ethernet*

Alimentación

Para alimentar la placa Arduino existen dos posibilidades, un convertidor AC-DC o una batería, basta con conectar el convertidor a la clavija de alimentación de la tarjeta y para utilizar la batería es necesario conectar los cables a los pines digitales Vin y GND de la placa.

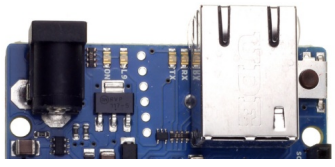

Figura 20. Toma de alimentación externa

La placa Arduino puede ser alimentada por una fuente de alimentación externa de 6 a 20 voltios. El rango recomendado es de 7 a 12 voltios para no dañar la placa y evitar el sobrecalentamiento.

Los pines correspondientes a la alimentación son:

- **Vin**: pin para la alimentación externa.
- **5V**: pin para alimentar el microcontrolador y otros componentes de la placa. Esta tensión puede provenir del pin Vin o de la conexión USB del ordenador al cual está conectada la placa.
- **3V3**: fuente de tensión generada por el chip FTDI. La corriente máxima es de 50 mA.
- **GND**: pin a tierra.
- **IOREF**: tensión de referencia para el microcontrolador.

Figura 21. Pines de alimentación

Memoria

El microcontrolador ATmega328 tiene 32 KB de memoria flash, 2 KB de SRAM y 1 KB de EEPROM.

Esta placa incluye una ranura para insertar una tarjeta de memoria micro SD, ya que el microcontrolador no posee mucho espacio para almacenar datos y existen algunos ejemplos en los que se necesita crear archivos de texto para almacenar la información que se recibe por el puerto serie.

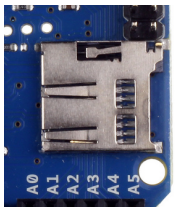

Figura 22.
Lector microSD

Entrada y salida

Hay 14 pines digitales que podemos utilizar como entrada o salida:

- **Serie: 0 (RX) y 1 (TX).** Pines que permiten recibir y transmitir información a través del puerto serie.
- **Interrupciones externas: 2 y 3.** Pines a través de los cuales se pueden enviar interrupciones a la placa.
- **PWM: 3, 5, 6, 9, y 10.** Pines de salida PWM.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** Comunicación SPI utilizando la biblioteca SPI.
- **LED: 9.** Este pin está asociado al LED integrado en la placa.
- **A0 - A5.** Estos seis pines son entradas analógicas.
- **TWI : A4 (SDA) y A5 (SCL).** Dichos pines permiten la comunicación TWI usando la librería Wire.
- **AREF.** Voltaje de referencia para las entradas analógicas.
- **Reset.** Pin para reiniciar el microcontrolador.

Figura 23. Pines de entrada y salida

Comunicación

Se puede comunicar la placa Arduino con el ordenador, otro Arduino, u otros microcontroladores. Los diferentes medios de comunicación son:

- El puerto serie con la ayuda de la biblioteca Software Serial Library.
- Comunicación SPI y TWI con las bibliotecas SPI y TWI.
- Puerto *Ethernet* (biblioteca *Ethernet*).
- Lector microSD (librería SD) .

Entorno Arduino

El entorno de Arduino está disponible en el sitio web oficial[5]. Existen diferentes versiones en función del sistema operativo con el que se trabaja. En este caso hemos utilizado la versión de Windows Arduino 1.4 para la realización del proyecto.

Este entorno está constituido de un editor de texto para escribir el código, una parte dedicada a los mensajes de error, un monitor serie, una barra de herramientas con los botones más prácticos y diferentes menús. Gracias a este software podemos cargar el código y comunicarnos con la placa Arduino.

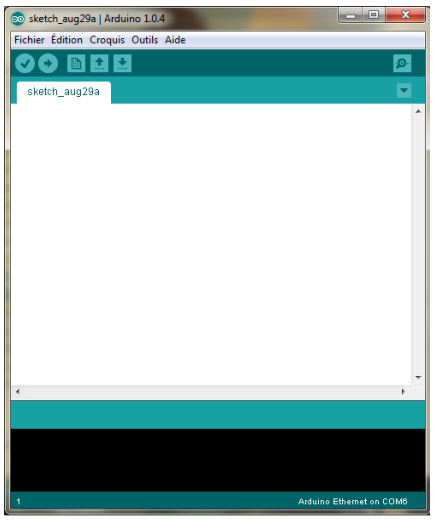

Figura 24. Entorno Arduino

[5] http://arduino.cc/es/Guide/Environment

Bibliotecas

Podemos encontrar una gran variedad de bibliotecas que nos permiten sacar el máximo partido a la placa.
En la web oficial, existen muchas bibliotecas que podemos descargar de forma gratuita.

En este caso particular, descargamos las siguientes bibliotecas para poder poner en práctica los ejemplos necesarios para llevar a cabo el presente proyecto:

- ***Ethernet***, para comunicarse con el microcontrolador Arduino a través del puerto *Ethernet*.
- **SD**, para escribir y leer los datos de la tarjeta de memoria SD.
- **SdFat,** para los intercambios con la memoria SD.
- **Tinkerkit**, funciones prediseñadas que permiten gestionar los componentes Tinkerkit.
- **xPL**, biblioteca que contiene los archivos para ejecutar el protocolo xPL en la placa Arduino.

Monitor serie

El entorno Arduino incluye un monitor serie para enviar o recibir datos. El monitor muestra los datos que se envían desde la placa Arduino. Para enviar información a la placa basta con escribir una línea de comandos en el monitor serie y hacer clic en “enviar”. Para que el monitor serie funcione correctamente es necesario determinar la velocidad (tasa en baudios) de intercambio entre la placa y el monitor serie. En nuestro caso, por defecto, utilizamos 9600.

Anexo II. Pulsador - Led

A continuación se presentan los códigos utilizados para la puesta en marcha del segundo escenario definido en el capítulo 5 del documento.

Código del microcontrolador Arduino emisor

```
/*
 Button

 Turns on and off a T010111 LED Module connected to O0,
 when pressing a T000180 pushbutton attached to I0.

 This example code is in the public domain.

 created in Dec 2011
 by Federico Vanzati

 based on  http://www.arduino.cc/en/Tutorial/Button
 */

// include the TinkerKit library
#include <TinkerKit.h>

// creating the object ‘button’ that belongs to the ‘TKButton’ class
// and giving the value to the desired input pin
TKButton button(I0);

// creating the object ‘led’ that belongs to the ‘TKLed’ class
// and giving the value to the desired output pin
TKLed led(O0);

void setup() {
  // TinkerKit ‘object’ eliminate the need for pin declaration with pinMode()
     Serial.begin(9600);
}

void loop()
{
  // check if the pushbutton is pressed

  if (button.get() == HIGH) {   // if it is, the button.state() is HIGH
    led.on();                   // turn LED on
    Serial.println('a');          // Enviamos 'a' por Tx (puerto serie)
    delay(1000);                // Retardo de 0,1 s
}
  else {                      // if it is not, the button.state() is LOW
    led.off();                // turn LED off
    Serial.println('b');          // Enviamos 'b' por Tx (puerto serie)
```

```
    delay(1000);                // Retardo de 0,1 s
}
}
```

Código del microcontrolador Arduino receptor

```
#include <TinkerKit.h>


TKLed led(O0);
int byte_entrada = 0;


void setup()
{
  Serial.begin(9600);    // Inicio el puerto serie a 9600 baudios
  led.on();
}

void loop()
{
  if (Serial.available()>0) {           // lee si al puerto Rx le llegan datos
    byte_entrada = Serial.read();  // almaceno en byte_entrada el dato que llega
al puerto
    Serial.print("J'ai recu : ");
    Serial.println(byte_entrada, DEC);
    if (byte_entrada == 'a')          // Si es una 'a'
    {
     led.off();
     delay(2000);
    }
    else //  if (byte_entrada == 'b')          // Si es una 'b'
    {
     led.on();
    }
  }

}
```

Anexo III. Servidor - Cliente

Para comprobar que nuestro programa funcionaba, verificamos escribiendo la dirección ip de la placa Arduino en la barra del navegador de internet y como vemos en la siguiente captura de pantalla, obtuvimos en el navegador Firefox el cliente que habíamos programado.

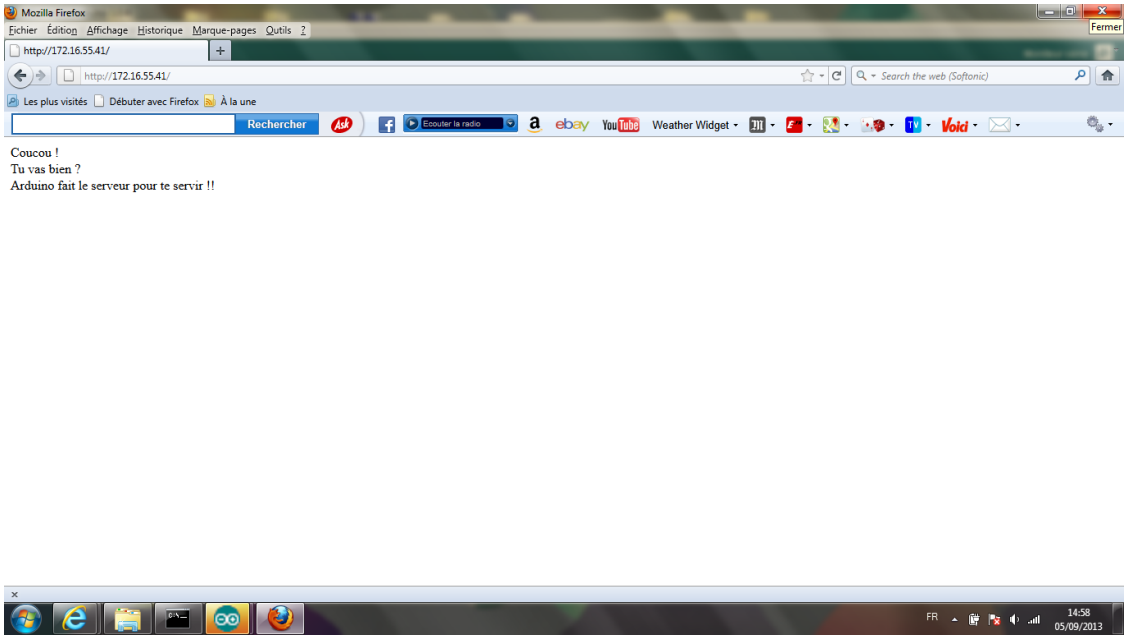

Figura 25. Navegador Firefox establecido como cliente

Y como podemos ver en el monitor serie del entorno Arduino, el proceso de petición de la creación del navegador como cliente sigue una evolución correcta como se establece en el código presentado en el siguiente apartado.

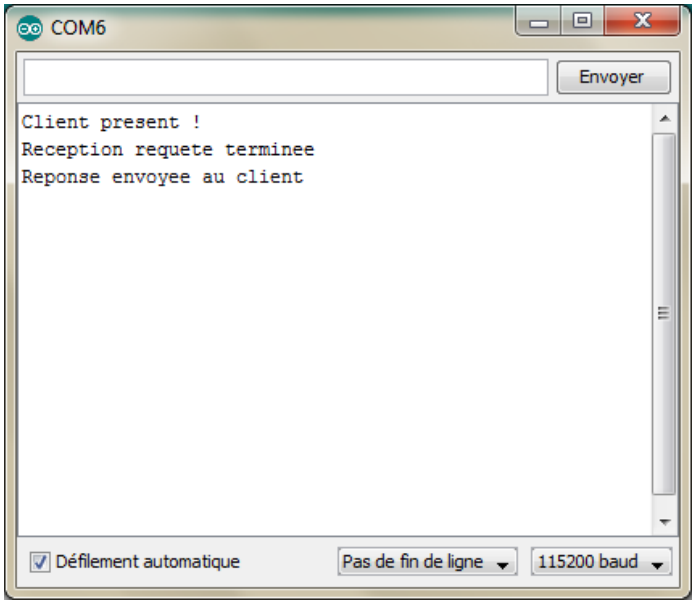

Figura 26. Verificación del proceso del programa

Código Arduino - *Ethernet*

```
// --- Programme Arduino ---
 // Copyright X. HINAULT - Créé le 03/10/2010
 // www.mon-club-elec.fr

//  Code sous licence GNU GPL :
//  This program is free software: you can redistribute it and/or modify
 //  it under the terms of the GNU General Public License as published by
 //  the Free Software Foundation, either version 3 of the License,
 //  or any later version.
 //  This program is distributed in the hope that it will be useful,
 //  but WITHOUT ANY WARRANTY; without even the implied warranty of
 //  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
the
 //  GNU General Public License for more details.
 //  You should have received a copy of the GNU General Public License
 //  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 //

// --- Que fait ce programme ? ---
 /* Ce programme configure le module Ethernet
 couplé à la carte Arduino en serveur HTTP
 On crée un réseau local avec le PC.

Ce programme envoie du code HTML basique
 vers un navigateur client (Firefox of course !)
 et affiche un texte simple dans le navigateur

Quelques messages témoins sont affichés dans le Terminal Série

*/

// --- Fonctionnalités utilisées ---

// --- Circuit à réaliser ---

//***************** Entête déclarative *******
 // A ce niveau sont déclarées les librairies incluses, les constantes, les
variables...

// --- Inclusion des librairies utilisées ---
 #include <SPI.h>
 #include <Ethernet.h>
 #include <Server.h>
 #include <Client.h>


// --- Déclaration des constantes ---
```

```
// --- constantes des broches ---

// --- Déclaration des variables globales ---

//--- déclaration du tableau d'adresse MAC ---
 byte mac[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xDA, 0x47 };

//---- tableau de l'adresse IP de la carte Arduino
 byte ip[] = { 172,16,55,41 }; // le PC a pour IP : 172.16.55.40

//----- tableau de l'adresse de la passerelle ---
byte passerelle[] = { 172,16,55,30 }; // l'adresse du PC de connexion

//----- tableau du masque de sous réseau
byte masque[] = { 255, 255, 255, 0 }; // idem masque sous-réseau du PC : 255.255.255.0


// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---

//--- création de l'objet serveur ----
 EthernetServer serveurHTTP(80); // crée un objet serveur utilisant le port 80 = port HTTP

//**************** FONCTION SETUP = Code d'initialisation *****
 // La fonction setup() est exécutée en premier et 1 seule fois, au démarrage du programme

void setup()   { // debut de la fonction setup()

// --- ici instructions à exécuter au démarrage ---

//---- initialise la connexion Ethernet avec l'adresse MAC, l'adresse IP et le masque
 Ethernet.begin(mac, ip, passerelle, masque);

//---- initialise le serveur ----
 serveurHTTP.begin();

//----initialise connexion série
Serial.begin(115200); // initialise connexion série à 115200 bauds
 // IMPORTANT : régler le terminal côté PC avec la même valeur de transmission

} // fin de la fonction setup()
 // ********************************************************************************

//*************** FONCTION LOOP = Boucle sans fin = coeur du programme *************
```

```
 // la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous tension

void loop(){ // debut de la fonction loop()

// --- ici instructions à exécuter par le programme principal ---

// crée un objet client basé sur le client connecté au serveur
   EthernetClient client = serveurHTTP.available();

  if (client) { // si l'objet client n'est pas vide
    // le test est VRAI si le client existe


    // message d'accueil dans le Terminal Série
       Serial.println ("Client present !");


    while (client.connected()) { // tant que le client est connecté

      if (client.available()) { // si des octets sont disponibles en lecture
       // le test est vrai si il y a au moins 1 octet disponible

     char c = client.read(); // lit l'octet suivant reçu du client (pour vider le buffer au fur à mesure !)


      } // --- fin client.available

     else { // si pas de caractères disponibles = requete client terminée

        // message Terminal Serie
         Serial.println ("Reception requete terminee");


        // envoi d'une entete standard de réponse http
           client.println("HTTP/1.1 200 OK");
           client.println("Content-Type: text/html");
           client.println();


          // affiche chaines caractères simples
            client.print("Coucou !");
            client.println("<br />"); // saut de ligne HTML
            client.print("Tu vas bien ? ");
            client.println("<br />"); // saut de ligne HTML
            client.print("Arduino fait le serveur pour te servir !! ");
            client.println("<br />"); // saut de ligne HTML
```

```
        // message Terminal Serie
         Serial.println ("Reponse envoyee au client");

        break; // on sort de la boucle while

       }


     } // --- fin while client connected

     // on donne au navigateur le temps de recevoir les données
      delay(1);

     // fermeture de la connexion avec le client
      client.stop();

   } //---- fin if client existe ----


} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin
 // ********************************************************************************

//*************** Autres Fonctions du programme *************

// --- Fin programme ---
```

Anexo IV. Interactuar gracias a Python

Lo que se pretendía en este ejemplo, era establecer la placa Arduino como servidor y el ordenador como cliente utilizando un menú interactivo creado gracias al lenguaje de programación Python.
Con este menú, se nos permite elegir entre las distintas opciones posibles como: encender el led, apagar el led o salir del programa. La siguiente imagen muestra el menú creado.

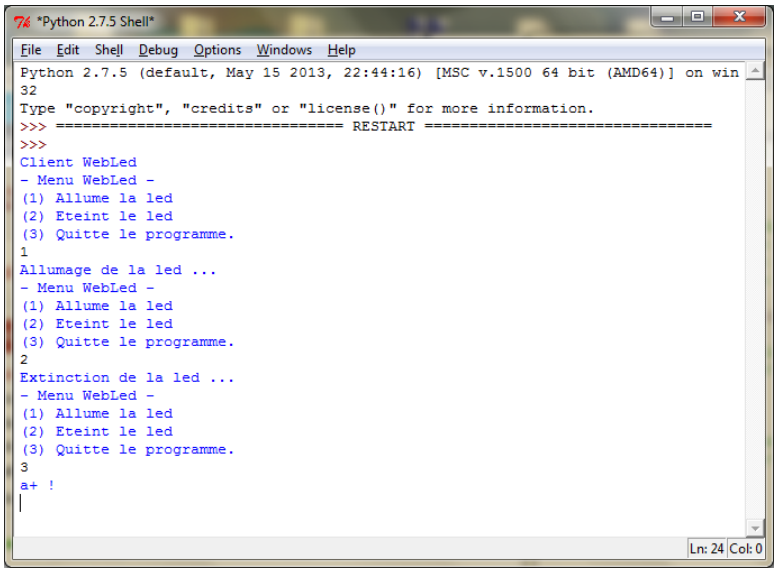

Figura 27. Menú creado con Python para interactuar con la placa Arduino

Gracias al monitor serie pudimos seguir el correcto funcionamiento de nuestro programa, ya que según la tecla que presionábamos en el teclado, se desencadenaba una acción u otra.

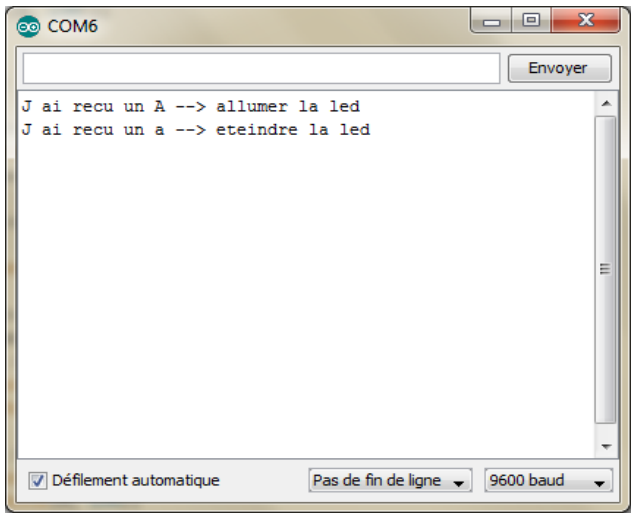

Figura 28. Monitor serie que muestra la correcta recepción de datos

Código servidor (placa Arduino)

```
#include <SPI.h>
#include <Ethernet.h>

/* Détails technique de la connexion ethernet */
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xDA, 0x47 };
byte ip[] = { 172,16,55,41 };

// Attachement d'un objet "server" sur le port 1337
EthernetServer server(1337);

void setup(){

// Configuration de la ethernet shield et du server
Ethernet.begin(mac, ip);
server.begin();
Serial.begin(9600);
// Mise en sortie de la broche avec notre led (par défaut éteinte)
pinMode(9, OUTPUT);
digitalWrite(9, LOW);}


void loop(){

// Attente de la connexion d'un client
EthernetClient client = server.available();
if (client && client.connected()) {

// si le client nous envoie quelque chose

if (client.available() > 0) {

// On regarde ce que le client nous demande
switch(client.read()){
  case 'A': // allumer la led
    Serial.println ("J ai recu un A --> allumer la led ");
    digitalWrite(9, HIGH);
    break;
  case 'a': // éteindre la led
    Serial.println ("J ai recu un a --> eteindre la led");
    digitalWrite(9, LOW);
    break;      }    }  }}
```

Código Cliente (ordenador)

```
import socket
print "Client WebLed"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("172.16.55.41", 1337))

q = 0
while q != 3:
	print "- Menu WebLed -"
	print "(1) Allume la led"
	print "(2) Eteint le led"
	print "(3) Quitte le programme."

	q = int(raw_input())
	if q == 1:
		print "Allumage de la led ..."
		s.send('A')
		continue

	if q == 2:
		print "Extinction de la led ..."
		s.send('a')
print "a+ !"
s.close()
raw_input()
```

Anexo V. Creación de una red LAN añadiendo un segundo servidor

Las diferencias de este ejemplo con respecto al anterior son sólo la declaración de un segundo servidor mediante las siguientes líneas de código:

```
>> s2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>> s2.connect(("172.16.55.42", 50322))
```

Servidor 1

```
#include <SPI.h>
#include <Ethernet.h>

/* Détails technique de la connexion ethernet */
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xDA, 0x47 };
byte ip[] = { 172,16,55,41 };

// Attachement d'un objet "server" sur le port 1337
EthernetServer server(1337);

void setup(){

// Configuration de la ethernet shield et du server
Ethernet.begin(mac, ip);
server.begin();

// Mise en sortie de la broche avec notre led (par défaut éteinte)
pinMode(9, OUTPUT);
digitalWrite(9, LOW);}

void loop(){

// Attente de la connexion d'un client
EthernetClient client = server.available();
if (client && client.connected()) {

// si le client nous envoie quelque chose

if (client.available() > 0) {

// On regarde ce que le client nous demande
switch(client.read()){
  case 'A': // allumer la led
    Serial.print ('J ai reçu un A');
    digitalWrite(9, HIGH);
    break;
  case 'E': // éteindre la led
```

```
    Serial.print ('J ai reçu un E');
    digitalWrite(9, LOW);
    break;      }    }  }}
```

Servidor 2

```
#include <SPI.h>
#include <Ethernet.h>

/* Détails technique de la connexion ethernet */
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xDB, 0x22 };
byte ip[] = { 172,16,55,42 };

// Attachement d'un objet "server" sur le port 1337
EthernetServer server(50322);

void setup(){

// Configuration de la ethernet shield et du server
Ethernet.begin(mac, ip);
server.begin();

// Mise en sortie de la broche avec notre led (par défaut éteinte)
pinMode(9, OUTPUT);
digitalWrite(9, LOW);}

void loop(){

// Attente de la connexion d'un client
EthernetClient client = server.available();
if (client && client.connected()) {

// si le client nous envoie quelque chose

if (client.available() > 0) {

// On regarde ce que le client nous demande
switch(client.read()){
  case 'a': // allumer la led
    Serial.print ('J ai reçu un a');
    digitalWrite(9, HIGH);
    break;
  case 'e': // éteindre la led
    Serial.print ('J ai reçu un e');
    digitalWrite(9, LOW);
    break;      }    }  }}
```

Menú Python

```
import socket
print ("Client WebLed")
s1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s1.connect(("172.16.55.41", 1337))
s2.connect(("172.16.55.42", 50322))

print (s1.getpeername())
print (s2.getpeername())
q = 0
while q != 5:
	print ("- Menu WebLed -")
	print ("(1) Allume la led carte 1")
	print ("(2) Allume la led carte 2")
	print ("(3) Eteint le led carte 1")
	print ("(4) Eteint le led carte 2")
	print ("(5) Quitte le programme.")

	q = int (input())
	if q == 1:
		data="A"
		print ("Allumage de la led carte 1...")
		s1.send(data.encode())
		continue

	if q == 2:
		data="a"
		print ("Allumage de la led carte 2...")
		s2.send(data.encode())

	if q == 3:
		data="E"
		print ("Extinction de la led carte 1...")
		s1.send(data.encode())

	if q == 4:
		data="e"
		print ("Extinction de la led carte 3...")
		s2.send(data.encode())

print ("a+ !")
s1.close()
s2.close()
```

Anexo VI. Uso de una tarjeta de memoria microSD

Para comprobar el correcto funcionamiento, creamos un fichero de texto que se almacena en la tarjeta micro SD para posteriormente poder ser leído. Leemos dicho contenido y lo mostramos en el monitor serie para verificar el funcionamiento del programa.

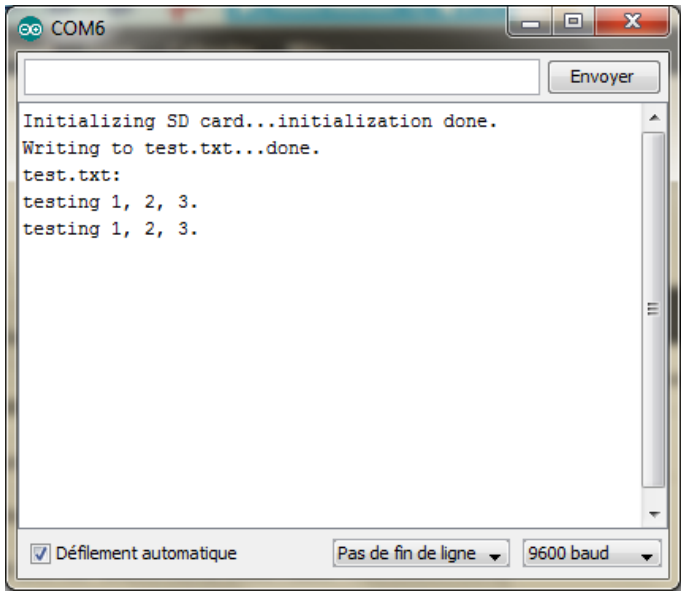

Figura 29. Creación y lectura de un fichero de texto

Código microSD lectura/escritura

```
/*
  SD card read/write

 This example shows how to read and write data to and from an SD card file
 The circuit:
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4

 created   Nov 2010
 by David A. Mellis
 modified 9 Apr 2012
 by Tom Igoe

 This example code is in the public domain.
```

```
 */

#include <SD.h>

 File myFile;

void setup()
{
  // Open serial communications and wait for port to open:
   Serial.begin(9600);
    while (!Serial) {
     ; // wait for serial port to connect. Needed for Leonardo only
   }


   Serial.print("Initializing SD card...");
   // On the Ethernet Shield, CS is pin 4. It's set as an output by default.
   // Note that even if it's not used as the CS pin, the hardware SS pin
   // (10 on most Arduino boards, 53 on the Mega) must be left as an output
   // or the SD library functions will not work.
    pinMode(10, OUTPUT);

   if (!SD.begin(4)) {
     Serial.println("initialization failed!");
     return;
   }
   Serial.println("initialization done.");

   // open the file. note that only one file can be open at a time,
   // so you have to close this one before opening another.
   myFile = SD.open("test.txt", FILE_WRITE);

   // if the file opened okay, write to it:
   if (myFile) {
     Serial.print("Writing to test.txt...");
     myFile.println("testing 1, 2, 3.");
     // close the file:
     myFile.close();
     Serial.println("done.");
   } else {
     // if the file didn't open, print an error:
     Serial.println("error opening test.txt");
   }

   // re-open the file for reading:
   myFile = SD.open("test.txt");
   if (myFile) {
     Serial.println("test.txt:");
```

```
    // read from the file until there's nothing else in it:
    while (myFile.available()) {
        Serial.write(myFile.read());
    }
    // close the file:
    myFile.close();
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }
}

void loop()
{
    // nothing happens after setup
}
```

Anexo VII. Protocolo domótico xPL

Tal y como se detalla en el documento, se presenta a continuación una captura de pantalla del programa Wireshark que muestra que la placa Arduino con dirección ip 172.16.55.41 envía cada 10 segundos un paquete del tipo *heartbeat*. La destinación de dichos paquetes es la dirección de *broadcast* de la red, es decir, que todos los equipos de la red en los que se ejecuta el protocolo xPL son destinatarios de estos paquetes. El contenido del paquete, recuadrado en rojo en la imagen, muestra el correcto envío de los paquetes emitidos según la definición del código.

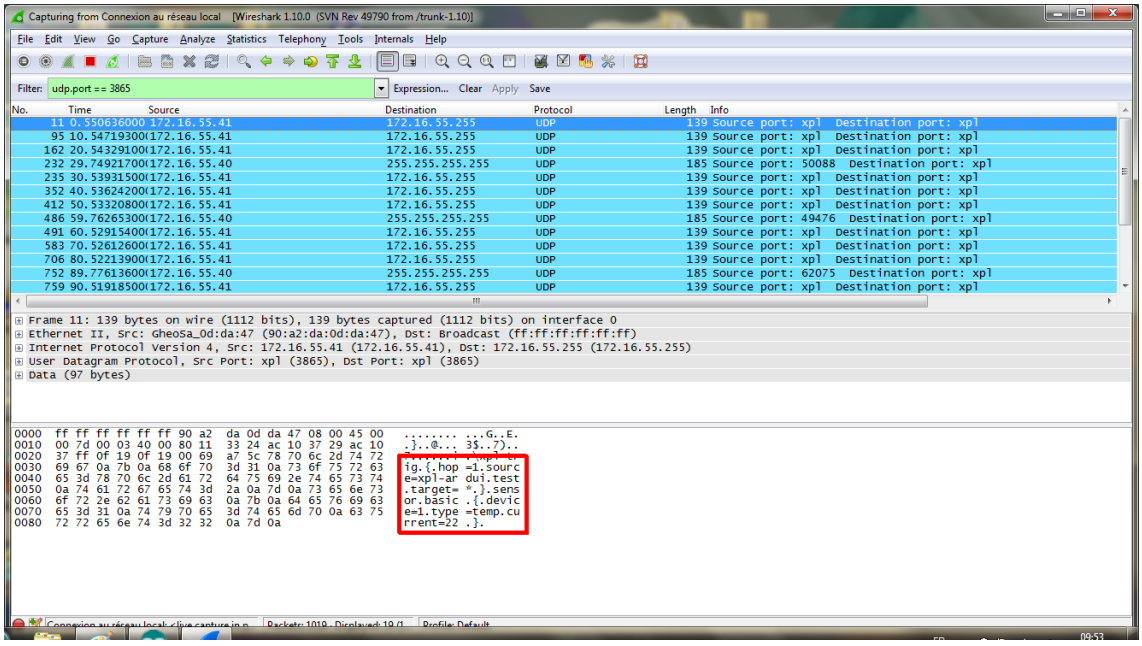

Figura 30. Definición del contenido de los paquetes enviados por la placa Arduino

Para ver las aplicaciones que están conectadas a la red xPL lanzamos xPLDiag, este programa nos permite además poder detectar los posibles fallos en la red xPL mostrándonos el origen de los mismos. El resultado de lanzar dicho programa nos muestra las siguientes aplicaciones activas en la red xPL.

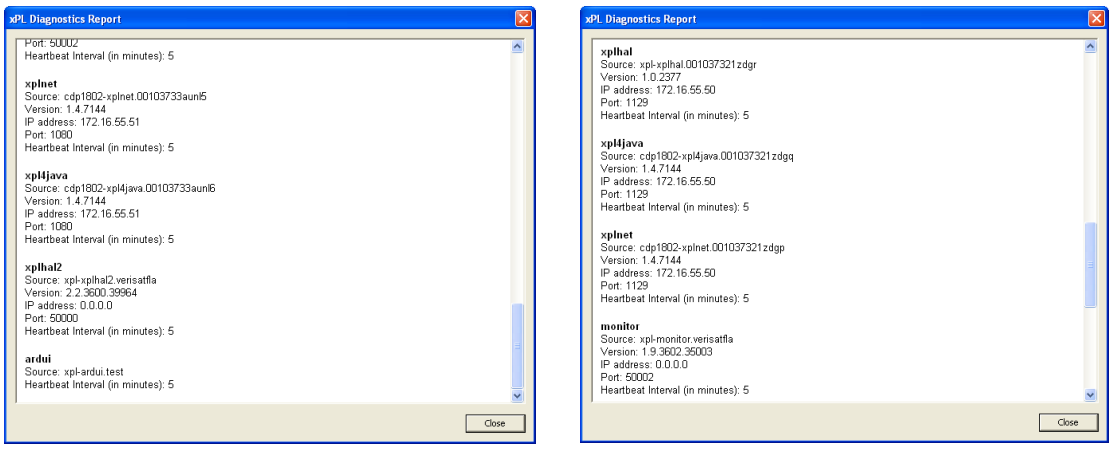

Figura 31. Aplicaciones xPL

La siguiente imagen muestra la creación de un paquete con el programa xPLHal Manager.

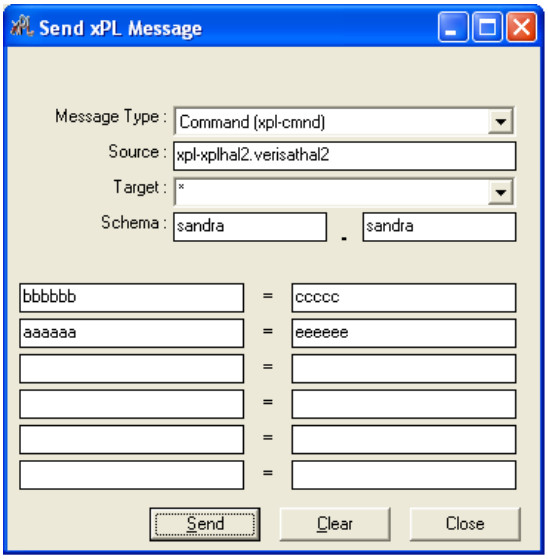

Figura 32. Creación y envío de un mensaje xPL

La siguiente ilustración corresponde a la citada captura de pantalla del programa Wireshark con la que se da por comprobado el correcto funcionamiento de la red xPL que instalamos en este escenario. Como se explica en el documento, el paquete recuadrado en rojo, demuestra que el *hub* xPL reenvía correctamente los paquetes a cada uno de los dispositivos existentes en la red.

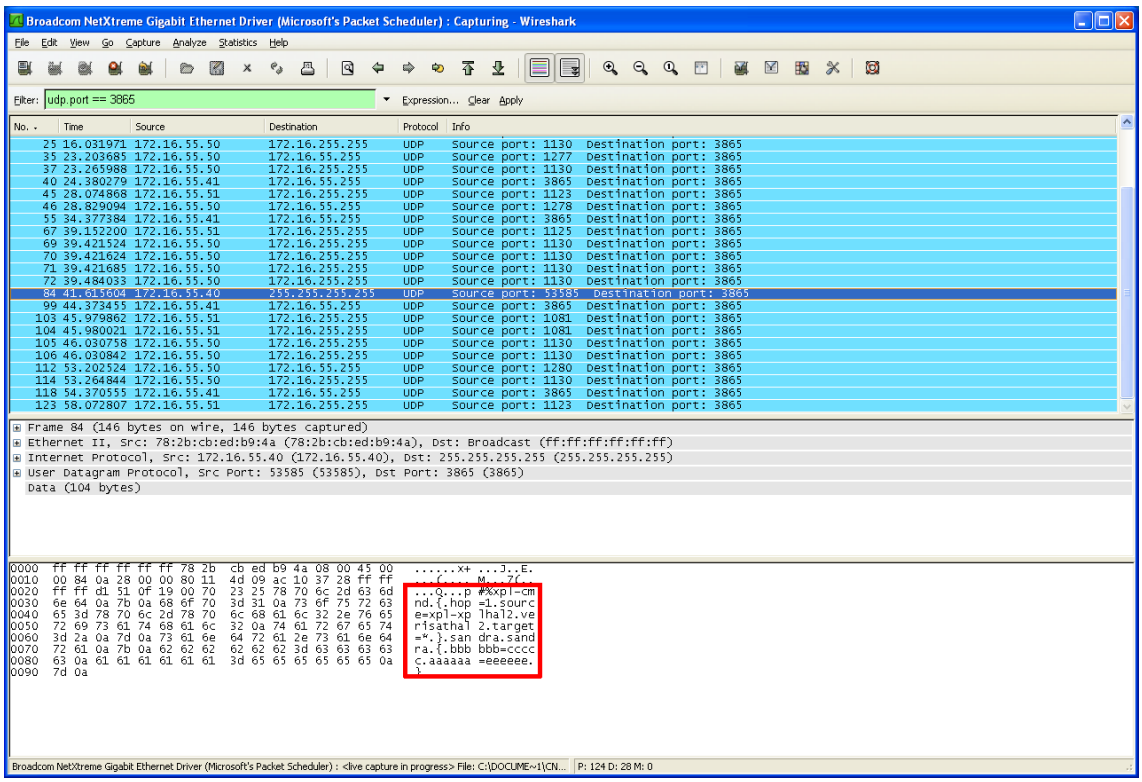

Figura 33. Captura del paquete que muestra el correcto funcionamiento del protocolo xPL

Como se comenta en el documento, la puesta en marcha de una red xPL es bastante caprichosa y requiere de los siguientes pasos para inicializarla y ponerla en marcha:

- Lanzar los *hubs* en todas las máquinas que están conectadas a la red para permitir el encaminamiento de todos los paquetes provenientes de cada aplicación y poder crear así la lista de los dispositivos presentes en cada máquina. Este método que utiliza el protocolo xPL permite establecer la red a medida que se van conectando los dispositivos.
- Lanzar xPLHal en el equipo con Windows XP para supervisar los paquetes que fluyen por la red, enviar paquetes, modificar la configuración de los dispositivos existentes ...
- Iniciar Wireshark para analizar los paquetes que se envían por la red.
- Iniciar xPLDiag en el ordenador xPL, puesto que el envío de paquetes proviene de esta aplicación, activando así el envío de todos los paquetes en la red. El envío de paquetes de paquetes *heartbeat* del programa xPLDiag permite la creación de tráfico de red, mediante el envío de las respuestas de todos los dispositivos existentes en la red.
- Envío, configuración y monitoreo de los paquetes a partir del programa xPLHal Manager.

Código xPL Arduino emisor

```
/*
 * xPL.Arduino v0.1, xPL Implementation for Arduino
 *
 * This code is parsing a xPL message stored in 'received' buffer
 * - isolate and store in 'line' buffer each part of the message -> detection of EOL character (DEC 10)
 * - analyse 'line', function of its number and store information in xpl_header memory
 * - check for each step if the message respect xPL protocol
 * - parse each command line
 *
 * Copyright (C) 2012 johan@pirlouit.ch, olivier.lebrun@gmail.com
 * Original version by Gromain59@gmail.com
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
*/

#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

#include "xPL.h"

xPL xpl;

unsigned long timer = 0;

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xDA, 0x47 };
IPAddress ip(172, 16, 55, 41);
IPAddress broadcast(172, 55, 16, 255);
EthernetUDP Udp;
```

```
void SendUdPMessage(char *buffer)
{
   Udp.beginPacket(broadcast, xpl.udp_port);
   Udp.write(buffer);
   Udp.endPacket();
}

void setup()
{
  Serial.begin(115200);
  Ethernet.begin(mac,ip);
  Udp.begin(xpl.udp_port);

  xpl.SendExternal = &SendUdPMessage;  // pointer to the send callback
  xpl.SetSource_P(PSTR("xpl"),  PSTR("arduino"),  PSTR("test"));  //  parameters
for hearbeat message
}

void loop()
{
  xpl.Process();  // heartbeat management

   // Example of sending an xPL Message every 10 second
   if ((millis()-timer) >= 10000)
   {
     xPL_Message msg;

     msg.hop = 1;
     msg.type = XPL_TRIG;

     msg.SetTarget_P(PSTR("*"));
     msg.SetSchema_P(PSTR("sensor"), PSTR("basic"));

     msg.AddCommand_P(PSTR("device"),PSTR("1"));
     msg.AddCommand_P(PSTR("type"),PSTR("temp"));
     msg.AddCommand_P(PSTR("current"),PSTR("22"));

     xpl.SendMessage(&msg);

     Serial.println("coucou");
     timer = millis();
   }
}
```

Código xPL Arduino analizador

```
/*
 * xPL.Arduino v0.1, xPL Implementation for Arduino
 *
 * This code is parsing a xPL message stored in 'received' buffer
 * - isolate and store in 'line' buffer each part of the message -> detection of EOL character (DEC 10)
 * - analyse 'line', function of its number and store information in xpl_header memory
 * - check for each step if the message respect xPL protocol
 * - parse each command line
 *
 * Copyright (C) 2012 johan@pirlouit.ch, olivier.lebrun@gmail.com
 * Original version by Gromain59@gmail.com
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
*/

#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

#include "xPL.h"

xPL xpl;

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xDB, 0x22 };
IPAddress ip(172, 16, 55, 42);
IPAddress broadcast(172, 16, 55, 255);
EthernetUDP Udp;

void SendUdPMessage(char *buffer)
{
   Udp.beginPacket(broadcast, xpl.udp_port);
```

```
    Udp.write(buffer);
    Udp.endPacket();
}

void AfterParseAction(xPL_Message * message)
{
          if (xpl.TargetIsMe(message))
    {
      if (message->IsSchema_P(PSTR("lighting"), PSTR("basic")))
      {
        Serial.println(PSTR("is lighting.basic"));
      }
    }

    // show message
    Serial.println(message->toString());
}

void setup()
{
  Serial.begin(115200);
  Ethernet.begin(mac,ip);
  Udp.begin(xpl.udp_port);

  xpl.SendExternal = &SendUdPMessage;  // pointer to the send callback
  xpl.AfterParseAction = &AfterParseAction;  // pointer to a post parsing action
callback
  xpl.SetSource_P(PSTR("xpl"), PSTR("arduino"), PSTR("test")); // parameters
for hearbeat message
}

void loop()
{
  xpl.Process();  // heartbeat management

  int packetSize = Udp.parsePacket();
  if(packetSize)
  {
          char xPLMessageBuff[XPL_MESSAGE_BUFFER_MAX];

    // read the packet into packetBufffer
    Udp.read(xPLMessageBuff, XPL_MESSAGE_BUFFER_MAX);

    // parse message
    xpl.ParseInputMessage(xPLMessageBuff);
  }
}
```